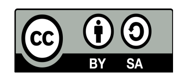
Lydia Pintscher (éd.)

Libres conseils

Logiciels libres et *open source*

Ce que nous aurions aimé savoir avant de commencer





Framasoft a été créé en novembre 2001 par Alexis Kauffmann. En janvier 2004, une association a vu le jour pour soutenir le développement du réseau. Pour plus d'information sur Framasoft, consulter http://www.framasoft.org.

Se démarquant de l'édition classique, les Framabooks sont dits « livres libres » parce qu'ils sont placés sous une licence qui permet au lecteur de disposer des mêmes libertés qu'un utilisateur de logiciels libres. Les Framabooks s'inscrivent dans cette culture des biens communs qui, à l'instar de Wikipédia, favorise la création, le partage, la diffusion et l'appropriation collective de la connaissance.

Le projet Framabook est coordonné par Christophe Masutti. Pour plus d'information, consultez http://framabook.org.

***

ISBN : 979-10-92674-04-0

Prix : 16 euros

Dépôt légal : octobre 2013, Framasoft (impr. *lulu.com*, Raleigh, USA)

Pingouins : LL de Mars, Licence Art Libre

Couverture : création originale par Nadège Dauvergne, Licence CC-By

Mise en page avec LibreOffice


Framabook
Le pari du livre libre

Préambule

Pour les géants et ceux qui se tiendront sur leurs épaules

Préface

Tristan Nitot

S’il est un livre que j’aurais aimé lire au début de mon aventure chez Mozilla en 1998, et quand nous avons monté Mozilla Europe en 2003, c’est bien celui-ci.

Depuis, j’ai pu rencontrer plusieurs de ses contributeurs de la communauté du Libre. J’ai longuement parlé avec eux de ces sujets, mais il était difficile de trouver du temps en commun et de se trouver au même endroit en même temps. J’ai aussi étudié comment ces projets Libres fonctionnent, et ce parfois depuis l’intérieur. J’utilise quotidiennement Wikipédia et participe dès que possible à Wikimedia Commons. J’ai échangé avec les gens de GPL Violations, relayé les actions de la FSFE. J’ai utilisé les logiciels conçus par ces communautés, de Samba à GNOME en passant par Red Hat Linux (ma première distribution, une 5.2, en 1998 !), Apache, StatusNet et Ubuntu. Une des auteurs de cet ouvrage, Selena, est même venue travailler chez Mozilla. Mais si j’avais eu une telle somme de savoir, recueillie dans un livre, qui plus est dans ma langue natale, quelle énergie j’aurais pu économiser ! Aujourd’hui, vous avez cette somme de savoir entre les mains, et elle n’a pas de prix.

Bien sûr, on pourra se référer au prix du livre physique, et dire que ça ne vaut qu'une poignée d'euros. On pourrait dire aussi que ça ne vaut rien, puisque le livre est téléchargeable gratuitement sur Internet. Mais sa valeur est bien plus grande. Marc Andreessen, le chef de l'équipe qui a créé l'un des premiers navigateurs web, Mosaic, avant de fonder Netscape, déclarait il y a quelques mois :

> « Le logiciel va dévorer le monde. »

Il a raison. En fait, le logiciel est déjà en train de dévorer le monde. Le logiciel touche un nombre croissant de nos actions quotidiennes. On apprend sur Wikipédia, on s'informe sur le Web, on papote sur les réseaux sociaux, on joue en ligne, on fait ses démarches administratives sur le Net, on cherche un plan de la ville sur OpenStreet Map, on regarde un film en VOD via une box ADSL. Le logiciel touche chaque aspect de nos vies, et ça n'est qu'un début.

Mais il y a quelque chose de fondamental que peu de gens ont bien compris : c'est celui qui développe le logiciel qui décide de ce que les utilisateurs peuvent ou ne peuvent pas faire avec ce logiciel. Il reste à décider si vous voulez être un consommateur passif ou bien un acteur de votre vie numérique. Être un client ou devenir un citoyen.

Ce livre est écrit par des passionnés qui ont décidé d'être des citoyens et des acteurs du numérique. Ils ont de plus décidé de permettre à d'autres de le devenir. Avec ce livre.

Alors ce livre, c'est bien davantage qu'un recueil de conseils. C'est un manuel pour permettre l'appropriation des outils numériques par les utilisateurs. C'est un manuel pour apprendre à construire le futur numérique que vous voulez, et pas celui que d'autres vous laisseront. Faites-en bon usage.

Avant-propos

Logiciels libres et *open source*
*C*e que nous aurions aimé savoir avant de commencer

Georg Greve

Georg Greve a fondé la Free Software Foundation Europe *(FSFE) en 2000 et en a été le président fondateur jusqu'en 2009. Durant cette période, il a été responsable du lancement et du développement de nombreuses activités de la FSFE, telles que les alliances, la politique ou les travaux juridiques. Il a intensivement travaillé avec de nombreuses communautés. Aujourd'hui, il poursuit ce travail en tant qu'actionnaire et PDG de Kolab Systems AG, une société qui se consacre entièrement aux logiciels libres. Pour ses actions en faveur du logiciel libre et des standards ouverts, Georg Greve a été décoré de la croix fédérale du mérite (*Bundesverdienstkreuz am Bande*) par la République fédérale d'Allemagne le 18 décembre 2009.*

Libres conseils[1] est une base de connaissances provenant d'une grande variété de projets de logiciels libres. Elle répond à des questions dont 42 contributeurs majeurs auraient aimé connaître les réponses lorsqu'ils ont débuté. Vous aurez ainsi une longueur d'avance quelle que soit la façon dont vous contribuez et quel que soit le projet que vous avez choisi.

Les projets de logiciels libres modifient le paysage du logiciel de façon impressionnante grâce à des utilisateurs dévoués et une

1 En anglais : *Open Advice* (http://open-advice.org).

gestion innovante. Chacun apporte quelque chose au mouvement à sa façon, avec ses capacités et ses connaissances. Cet engagement personnel et la puissance du travail collaboratif sur Internet donnent toute leur force aux logiciels libres et c'est ce qui a rassemblé les auteurs de ce livre.

Ce livre est la réponse à la question : « Qu'auriez-vous aimé savoir avant de commencer à contribuer ? ». Les auteurs offrent un aperçu de la grande variété de talents qu'il faut rassembler pour réussir un projet de logiciel : le codage bien sûr, mais aussi le design, la traduction, le marketing et bien d'autres compétences. Nous sommes là pour vous donner une longueur d'avance si vous êtes nouveau. Et si ça fait déjà un moment que vous contribuez, nous sommes là pour vous donner un aperçu d'autres domaines et projets.

Ce livre parle de communauté et de technologies. Il est le fruit d'un travail collectif, un peu comme la technologie que nous construisons ensemble. Si c'est votre première rencontre avec notre communauté, vous pourrez trouver étrange de penser qu'une communauté puisse être le moteur qui propulse la technologie. La technologie n'est-elle pas l'œuvre des grands groupes industriels ? En fait, pour nous c'est presque l'inverse. Les auteurs de ce livre sont tous membres de ce que vous pourriez appeler la communauté du logiciel libre. Un groupe de personnes qui partagent l'idée fondatrice que les logiciels sont plus puissants, plus utiles, plus flexibles, mieux contrôlables, plus justes, plus englobants, plus durables, plus efficaces, plus sûrs et finalement simplement meilleurs quand ils sont fournis avec les quatre libertés fondamentales : la liberté d'utiliser, la liberté d'étudier, la liberté de partager et la liberté d'améliorer le logiciel.

Et bien qu'il y ait maintenant un nombre croissant de communautés qui ont appris à se passer de la proximité géographique grâce aux moyens de communication virtuels, c'est cette communauté qui en a été le précurseur.

En fait, Internet et la communauté du logiciel libre[1] suivaient des développements mutuellement dépendants. Au fur et à mesure qu'Internet grandissait, notre communauté pouvait grandir en même temps. Mais sans les valeurs ni la technologie qu'apportait notre communauté, il ne fait aucun doute à mes yeux que jamais Internet n'aurait pu devenir ce réseau global reliant les personnes et les groupes du monde entier.

À ce jour, nos logiciels font fonctionner la majeure partie d'Internet, et vous devez en connaître au moins quelques-uns, comme Mozilla Firefox, OpenOffice.org, Linux, et peut-être même Gnome ou KDE. Mais notre technologie peut aussi se cacher dans votre téléviseur, votre routeur sans fil, votre distributeur automatique de billets, et même votre radio, système de sécurité ou bataille navale. Elle est littéralement omniprésente.

Ils ont été essentiels dans l'émergence de quelques-unes des plus grandes sociétés que vous connaissez, comme Google, Facebook, Twitter et bien d'autres. Aucune d'entre elles n'aurait pu accomplir autant en si peu de temps sans le pouvoir du logiciel libre qui leur a permis de monter sur les épaules de ceux qui étaient là avant eux.

Mais il existe également de nombreuses petites entreprises qui vivent de, avec, et pour le logiciel libre, dont la mienne, Kolab Systems. Le fait d'agir activement avec la communauté et dans un bon esprit est devenu un élément de succès essentiel pour nous tous.

Et c'est aussi vrai pour les plus grosses, comme Oracle nous l'a involontairement démontré durant et après sa prise de contrôle de Sun Microsystems. Il est important de comprendre que notre communauté n'est pas opposée au commerce. Nous aimons notre travail, et beaucoup d'entre nous en ont fait leur métier pour

1 Note de l'auteur : pour moi, l'*open source* n'est que l'un des aspects de cette communauté. Cet aspect particulier a trouvé son expression en 1998, c'est-à-dire quelque temps après l'arrivée d'Internet. Mais n'hésitez pas à lire « *open source* » au lieu de « logiciel libre » si vous préférez ce terme.

gagner leur vie et rembourser leurs crédits. Donc quand nous parlons de communauté, nous voulons dire des étudiants, des entrepreneurs, des développeurs, des artistes, des documentalistes, des professeurs, des bricoleurs, des hommes d'affaires, des commerciaux, des bénévoles et des utilisateurs.

Oui, des utilisateurs. Même si vous ne vous en êtes pas encore rendu compte ou n'avez jamais appartenu à une communauté, vous faites en réalité déjà partie de la nôtre. La question est de savoir si vous allez y participer activement. Et c'est cela qui nous différencie des poids lourds de la monoculture, des communautés fermées, des jardins clôturés de sociétés telles qu'Apple, Microsoft et d'autres. Nos portes sont ouvertes. Tout comme nos conseils. Et également notre potentiel. Il n'y a pas de limite à ce que vous pouvez devenir – cela dépend uniquement de votre choix personnel comme cela a été le cas pour chacun d'entre nous.

Donc si vous ne faites pas encore partie de notre communauté, ou si vous êtes simplement curieux, ce livre offre un bon point de départ. Et si vous êtes déjà un participant actif, ce livre pourrait vous offrir un aperçu de quelques facettes et de quelques perspectives qui seront nouvelles pour vous.

En effet, ce livre contient de précieuses graines de ce savoir implicite que nous avons l'habitude de construire et de transférer à l'intérieur de nos sous-communautés qui travaillent sur diverses technologies. Ce savoir circule généralement des contributeurs les plus expérimentés vers les moins expérimentés. C'est pourquoi il semble tellement évident et naturel à ceux qui fréquentent notre communauté.

Ce savoir et cette culture de la collaboration nous permettent de créer d'extraordinaires technologies avec de petites équipes du monde entier au-delà des différences culturelles, linguistiques et de nationalité. Cette manière de fonctionner permet de surpasser

des équipes de développement bien plus grandes de certaines des plus grosses sociétés au monde.

Tous les contributeurs de ce livre ont une expérience solide dans au moins un domaine, parfois plusieurs. Ils sont devenus des enseignants et des mentors. Au cours des quinze dernières années, j'ai eu le plaisir d'apprendre à connaître la plupart d'entre eux, de travailler avec beaucoup, et j'ai le privilège de compter certains parmi mes amis.

Comme l'a dit judicieusement Kévin Ottens pendant le Desktop Summit 2011 à Berlin, « construire une communauté, c'est construire de la famille et de l'amitié ».

C'est donc en réalité avec un profond sentiment de gratitude que je peux dire qu'il n'y a aucune autre communauté dont je préférerais faire partie, et je suis impatient de vous rencontrer à l'une ou l'autre des conférences à venir.

– Zürich, Suisse, le 20 août 2011

Merci !

Ce livre n’aurait pu voir le jour sans la participation de chaque auteur et des personnes suivantes, qui ont aidé à sa réalisation :

Anne Gentle (relecture), Bernhard Reiter (relecture), Celeste Lyn Paul (relecture), Daniel Molkentin (mise en page), Debajyoti Datta (site internet), Irina Rempt (relecture), Jeff Mitchell (relecture), Mans Rullgard (relecture), Noirin Plunkett (relecture), Oregon State University Open Source Lab (hébergement du site internet), Stuart Jarvis (relecture), Supet Pal Singh (site internet), Saransh Sinha (site internet), Vivek Prakash (site internet), Will Kahn-Greene (relecture)

Idées et innovations

1. Du code avant toute chose

Armijn Hemel

Armijn Hemel utilise des logiciels libres depuis 1994, lorsque son frère est revenu à la maison avec une pile de disquettes contenant l'une des premières versions de FreeBSD. Un an après, il migrait vers GNU/Linux et depuis, il n'utilise plus que des systèmes de type Unix, que ce soit chez lui, pour ses études à l'université d'Utrecht ou au travail. Depuis 2005, Armijn est membre du noyau dur de l'équipe de gpl-violations.org tout en possédant son propre cabinet de conseil (Tjaldur Software Governance Solutions) spécialisé dans la détection et la résolution de litiges nés de violations de licences GPL.

En 1999, je faisais tout juste mes premiers pas comme activiste dans le logiciel libre et *open source*. J'utilisais déjà Linux et FreeBSD depuis un certain nombre d'années, mais je n'étais encore qu'un simple utilisateur et je souhaitais apporter une contribution en retour. De mon point de vue, la meilleure manière de le faire était d'écrire du code. Étant donné que je ne trouvais aucun projet existant dans lequel j'aurais été à l'aise pour travailler, j'ai décidé de commencer mon propre projet. Avec le recul, je constate que plusieurs raisons m'ont poussé à débuter ce projet. L'une tenait à mes doutes sur la qualité de mon code : était-il assez bon pour être accepté dans un projet existant – je n'étais pas un programmeur brillant et d'ailleurs je ne le suis

toujours pas ? Pour un projet personnel, la question ne se pose pas. La seconde raison est l'arrogance de la jeunesse.

Mon idée était de créer un logiciel de présentation, qui pourrait imiter la plupart des fonctionnalités avancées – ou si vous préférez, les plus pénibles – de PowerPoint. À ce moment-là, OpenOffice.org n'existait pas encore et le choix était relativement limité : LaTeX et MagicPoint, qui sont davantage orientés vers le contenu textuel que sur les effets d'animation. Je voulais créer un logiciel multi-plateforme et, à l'époque, j'ai pensé que Java serait le meilleur choix pour le réaliser. Mon plan était donc de faire un logiciel de présentation, écrit en Java, qui aurait intégré tous ces effets animés. Je me suis décidé et j'ai commencé le projet.

Toute l'infrastructure nécessaire était en place : une liste de diffusion, un site web, un système de gestion de versions (CVS[1]). Mais ce qui n'existait pas encore, c'est le code source qui aurait permis à des contributeurs potentiels de travailler directement. La seule chose que j'avais à proposer, c'était quelques idées de ce que je voulais faire, un truc qui me démangeait, et des mots accrocheurs. En fait, je voulais que les gens me rejoignent pour créer le programme et que celui-ci devienne réellement un projet collaboratif.

J'ai commencé par concevoir des modèles – avec mes nouvelles connaissances en UML[2] – et les faire circuler. Rien ne s'est passé. J'ai essayé d'impliquer des contributeurs, mais créer une architecture de manière collaborative est très difficile – sans compter qu'*a priori*, ce n'est sûrement pas le meilleur moyen de créer un logiciel. Après un certain temps, j'ai laissé tomber et le projet est mort en silence, sans qu'une seule ligne de code ait été écrite. Chaque mois je recevais des messages par la liste de diffusion qui me rappelaient que ce projet avait un jour existé, j'ai donc demandé sa mise hors ligne.

1 CVS : Concurrent Version Systems, un système de gestion de versions concurrentes, très utilisé dans le logiciel libre.
2 UML : Unified Modeling Language, un langage de modélisation graphique.

J’en ai tiré une leçon précieuse, quoiqu’un peu douloureuse : dès lors que vous annoncez quelque chose et que vous souhaitez que les gens s’impliquent dans votre projet, assurez-vous au moins qu’il y ait un minimum de code disponible. Peu importe qu’il ne soit pas complètement terminé ; ce n’est pas grave s’il est mal dégrossi – au début en tout cas. Mais montrez au moins qu’il y a une base sur laquelle des contributeurs peuvent travailler et ainsi l’améliorer ; sans quoi, votre projet finira de la même manière que tant d’autres, dont le mien : aux oubliettes.

J’ai fini par trouver mon créneau pour contribuer au progrès du logiciel libre et *open source*, en veillant à la solidité de leurs fondements juridiques au sein du projet gpl-violations.org. Avec le recul, je n’ai jamais utilisé les effets animés dans les logiciels de présentation – et cela ne me manque pas d’ailleurs. En fait, je les trouve de plus en plus irritants : ils nous distraient trop du contenu. Pour faire mes présentations, je suis un utilisateur heureux de LaTeX Beamer[1] et occasionnellement – mais avec moins de plaisir – d’OpenOffice.org/LibreOffice.

1 https://fr.wikipedia.org/wiki/Beamer.

2. Tous les autres pourraient avoir tort, mais c’est peu probable

Evan Prodromou

Evan Prodromou est le fondateur de Wikitravel[1]*, StatusNet*[2] *et du réseau social* open source *Identi.ca*[3]*. Il contribue aux logiciels* open source *depuis 15 ans en tant que développeur, écrit de la documentation et se distingue à l’occasion comme agitateur. Il vit à Montréal, au Québec.*

La caractéristique la plus importante du fondateur d’un projet *open source*, dans les premières semaines ou premiers mois avant de lancer son logiciel dans le vaste monde, c’est une obstination de tête de mule face à l’écrasante évidence des faits. Si votre logiciel est si important, pourquoi personne ne l’a-t-il déjà écrit ? Peut-être que ce n’est même pas possible. Peut-être que personne d’autre que vous n’a besoin de ce que vous êtes en train de faire. Peut-être que vous n’êtes pas assez bon pour le faire. Peut-être que quelqu’un l’a déjà fait et que vous n’êtes simplement pas assez malin pour le trouver avec Google.

1 http://wikitravel.org/fr/Accueil.
2 http://status.net.
3 https://identi.ca.

Garder la foi à travers cette longue et sombre nuit est difficile ; seules les têtes de cochons opiniâtres et bornées peuvent y parvenir. Et vous voilà maintenant sur le champ de bataille des opinions les plus férocement défendues parmi les programmeurs. Quel est le meilleur langage de programmation à utiliser ? L'architecture de l'application ? Les standards d'écriture du code ? La licence du logiciel ? Le système de gestion de version ? Si vous êtes seul à travailler (ou à connaître !) le projet, vous devez en décider, de façon unilatérale.

Quand vous finissez par le lancer, cependant, cette détermination bornée et cette forte opinion ne jouent plus à votre avantage mais à votre détriment. Une fois que vous avez lancé le projet, vous aurez besoin de compétences diamétralement opposées pour faire des compromis et permettre à votre logiciel d'être plus utile aux autres. Et beaucoup de ces compromis vous sembleront vraiment mauvais.

Il est difficile d'accepter les contributions d'« étrangers » (c'est-à-dire des personnes qui ne sont pas vous). D'abord parce qu'ils se focalisent sur des choses vraiment triviales et sans importance : votre convention de nommage des variables par exemple, ou l'emplacement de certains boutons. Ensuite parce qu'ils ont invariablement tort. Après tout, si ce que vous avez fait n'avait pas été la bonne manière de faire, vous ne l'auriez pas fait ainsi dès le départ. Si votre façon de faire n'était pas la bonne, pourquoi votre code serait-il si populaire ?

Mais « mauvais » est relatif. Si faire un « mauvais » choix est ce qui rend votre logiciel plus accessible aux utilisateurs finaux, aux développeurs en aval, aux administrateurs ou aux empaqueteurs, est-ce que ce n'est pas en définitive un *bon* choix ?

La nature de ce genre de commentaires et de contributions est généralement négative. Les retours de la communauté sont essentiellement des réactions, ce qui implique qu'elles sont principalement des critiques. Avez-vous déjà rapporté un bogue qui disait

« J'aime beaucoup l'organisation du module hashtable.c » ou « Bravo d'avoir supprimé ce sous-sous-sous-menu » ? Les gens font un retour d'expérience car ils n'aiment pas la façon dont fonctionne votre logiciel à un instant T. Et ils ne sont pas toujours très diplomates à ce moment-là.

Il est difficile de répondre de façon positive à ce genre de retour. Nous engueulons parfois les commentateurs sur nos listes de diffusions en développement, ou fermons les rapports de bogues avec un rictus et un WONTFIX[1]. Pire encore, nous nous retirons dans notre tour d'ivoire, ignorant les suggestions externes ou les retours d'expérience, et nous nous blottissons contre notre confortable code qui sied si parfaitement à nos idées préconçues et à nos partis pris.

Si le logiciel est exclusivement destiné à votre usage, vous pouvez garder le code source et les infrastructures qui l'entourent comme terrain de jeu personnel. Mais si vous voulez que votre logiciel soit utilisé, qu'il compte pour d'autres personnes, qu'il change (peut-être) le monde, alors il va falloir que vous bâtissiez une saine et solide communauté d'utilisateurs, de contributeurs principaux, d'administrateurs et de développeurs de modules. Toutes ces personnes doivent avoir le sentiment que le logiciel leur appartient, aussi bien qu'à vous.

Il est difficile de se rappeler que chacune de ces voix dissidentes n'est jamais que la partie émergée de l'iceberg. Imaginez tous ces gens qui entendent parler de votre logiciel sans jamais prendre le temps de l'essayer. Ceux qui le téléchargent mais ne l'installent jamais. Ceux qui l'installent, restent bloqués, et l'abandonnent en silence. Et ceux qui veulent vous faire un retour, mais qui ne trouvent pas le système de rapports de bogues, les listes de diffusions pour les développeurs, les canaux IRC ou adresses personnelles de messagerie. Étant donnés les obstacles à la transmission d'un message, pour chaque personne qui parvient

1 WONTFIX : peut se traduire NE-SERA-PAS-RÉSOLU, mot-clé employé en réponse à un rapport de bogue, pour indiquer qu'il ne sera pas pris en considération.

à faire passer le sien, il y en a probablement une centaine qui souhaiterait des modifications. Il est donc essentiel d'être à l'écoute de ces voix quand elles parviennent jusqu'à vous.

Le responsable de projet est chargé de maintenir la vision et la finalité du logiciel. Nous ne pouvons tergiverser, aller d'avant en arrière sur la base de tel ou tel courriel d'utilisateur pris au hasard. Si un principe fondamental est en jeu, alors, bien sûr, il est important d'en maintenir la stabilité. Personne d'autre que le responsable de projet ne peut le faire.

Mais nous devons réfléchir : existe-t-il des questions non fondamentales qui puissent rendre le logiciel plus accessible, plus facile d'utilisation ? En fin de compte, notre travail se mesure à la manière dont nous touchons les utilisateurs, à la façon dont notre logiciel est utilisé et à l'usage qui en est fait. À quel point notre idée personnelle de ce qui est « bien » est-elle importante pour le projet et pour la communauté ? À quel point relève-t-elle uniquement de ce que le responsable aime, personnellement ? Si ce genre de problèmes non essentiels existent, alors il faut arrondir les angles, répondre aux demandes, et opérer des changements. Le projet n'en sera que meilleur pour tout le monde.

Recherches

3. Hors du labo, au grand air

La croissance des communautés *open source* autour des projets universitaires

Markus Kroëtzsch

Markus Kroëtzsch est post-doctorant au Département des Sciences Informatiques de l'Université d'Oxford. *Il a soutenu son doctorat en 2010 à l'Institut d'Informatique Appliquée et Méthodes Formelles de description du* Karlsruhe Institute of Technology. *Ses recherches portent sur le traitement automatique de l'information, depuis les fondements de la représentation de la connaissance formelle jusqu'à leurs domaines d'application, tel le Web sémantique. Il est le développeur principal de la plate-forme* Semantic MediaWiki[1]*, application de Web sémantique, co-éditeur des spécifications* W3C OWL2[2]*, administrateur principal du portail communautaire* semanticweb.org[3]*, et co-auteur de l'ouvrage* Foundations of Semantic Web Technologies[4].

Au sein des universités, les chercheurs développent de grandes quantités de logiciels, que ce soit pour valider une hypothèse, pour illustrer une nouvelle approche, ou tout simplement comme outil en appui à une étude. Dans la plupart des cas, un petit proto-

1 http://semantic-mediawiki.org.
2 http://www.w3.org/TR/owl2-overview.
3 http://semanticweb.org/wiki/Main_Page.
4 http://www.semantic-web-book.org/page/Foundations_of_Semantic_Web_Technologies

type dédié fait le travail, et il est déployé rapidement tandis que l'enjeu de la recherche évolue. Cependant, de temps à autre, une nouvelle approche ou une technologie émergente a le potentiel de changer complètement la manière de résoudre un problème. Cela promet de la réputation professionnelle, du succès commercial et la satisfaction personnelle d'amener une nouvelle idée à son plein potentiel. Le chercheur qui a fait une découverte de ce genre est alors tenté d'aller au-delà du prototype vers un produit qui sera réellement utilisé. Il est alors confronté à une toute nouvelle série de problèmes pratiques.

La peur de l'utilisateur

Dans l'un de ses célèbres essais sur l'ingénierie logicielle, Frederick P. Brooks Jr. permet de se faire une bonne idée des efforts liés à la maintenance d'un vrai logiciel et nous met en garde contre l'utilisateur :

> « Le coût total de la maintenance d'un programme largement utilisé est habituellement de 40 % ou plus de son coût de développement. De façon surprenante, ce coût est fortement influencé par le nombre d'utilisateurs. Plus il y a d'utilisateurs, plus il y a de bogues. »[1]

Bien que les chiffres soient probablement différents dans le contexte actuel, la remarque reste fondamentalement vraie. Elle pourrait même avoir été confirmée par l'usage généralisé de la communication instantanée. Pire encore : davantage d'utilisateurs ne produisent pas seulement davantage de bogues débusqués ; en général, ils expriment aussi plus de demandes. Qu'il s'agisse d'une véritable erreur, d'une demande de fonctionnalité, ou tout simplement d'une mauvaise compréhension du fonctionnement du logiciel, les demandes de l'utilisateur lambda ne ressemblent en rien à un rapport de bogue précis et technique. Et chaque

1 Frederick P. Brooks Jr.: *The Mythical Man-Month*. Essays on Software Engineering. Anniversary Edition. Addison-Wesley, 1995.

demande requiert l’attention des développeurs et occupe un temps précieux qui n’est plus disponible pour écrire du code.

Avec son esprit d’analyse, le chercheur anticipe ce problème. Dans sa lutte naturelle pour éviter un avenir sombre au service client, il peut carrément commencer à avoir *peur de l’utilisateur*. Dans le pire des cas, cela peut le mener à prendre des décisions qui vont à l’encontre du projet dans son ensemble ; sous des formes plus légères, cela peut tout de même mener le chercheur à cacher des produits logiciels brillants à ses utilisateurs potentiels. Plus d’une fois, j’ai entendu des chercheurs dire : « Nous n’avons pas besoin de plus de visibilité : nous recevons déjà suffisamment de courriels ! » Il est vrai que parfois l’investissement en communication que nécessite un outil logiciel va au-delà de ce qu’un chercheur peut y consacrer sans renoncer à son activité principale.

Pourtant, cette issue tragique aurait bien souvent pu être évitée. Brooks pouvait difficilement l’anticiper quand il a écrit ses essais. À l’époque, les utilisateurs étaient effectivement des clients et la maintenance logicielle faisait partie du produit qu’ils achetaient. Il fallait trouver un équilibre entre l’effort de développement, la demande du marché et le prix. De nos jours, c’est toujours le cas pour de nombreux produits logiciels commerciaux mais n'a que peu de rapport avec la réalité du développement à petite échelle de l’*open source*. Les utilisateurs habituels de l’*open source* ne paient pas pour le service qu’ils reçoivent. Leur attitude n’est donc pas celle d’un client exigeant, mais bien plus souvent celle d’un adepte enthousiaste et reconnaissant. Transformer cet enthousiasme en un soutien bienvenu ne compte pas pour rien dans l’art de réussir la maintenance d’un logiciel *open source* : l’intérêt croissant de l’utilisateur doit aller de pair avec une contribution croissante.

Reconnaître que les utilisateurs de logiciels *open source* ne sont pas que « des consommateurs qui ne paient pas » est une notion importante. Mais cela ne doit pas mener à surestimer leur

potentiel. Le pendant optimiste de la peur irrationnelle de l'utilisateur est la croyance que des communautés actives croissent naturellement avec pour seule base la licence choisie pour publier le code. Cette grave erreur de jugement est bizarrement toujours aussi commune et a scellé le destin de bien des tentatives de création de communautés ouvertes.

Semer et récolter

Le pluriel d'« utilisateur » n'est pas « communauté ». Si les premiers peuvent voir leur nombre s'accroître, la seconde ne grandit pas d'elle-même, ou alors elle grandit sans direction et surtout sans fournir le soutien espéré au projet. La mission du responsable de projet qui cherche à profiter de l'énergie brute des utilisateurs ressemble à celle d'un jardinier qui doit préparer un terrain fertile, planter et arroser les semis, et peut-être élaguer les pousses non désirées avant de pouvoir récolter les fruits. Par rapport aux récompenses qui viendront, l'investissement global est minime, mais il est essentiel de faire les bonnes choses, au bon moment.

Préparer le support technologique

Créer une communauté commence avant même que le premier utilisateur n'apparaisse. D'emblée, le choix du langage de programmation va déterminer le nombre de personnes qui pourront déployer et déboguer notre code. Objective Caml est sans doute un beau langage mais si l'on utilise plutôt Java, la quantité d'utilisateurs et de contributeurs potentiels augmentera de plusieurs ordres de grandeur. Les développeurs doivent donc faire des compromis puisque la technologie la plus répandue est rarement la plus performante ou la plus élégante. Cela peut être une démarche particulièrement difficile pour des chercheurs qui privilégient souvent la supériorité formelle du langage. Quand je travaillais sur Semantic MediaWiki, on m'a souvent demandé pourquoi nous utilisions PHP alors que Java côté serveur serait

tellement plus propre et performant. Comparons la taille de la communauté de Semantic MediaWiki et les efforts que demanderait le même développement basé sur du Java : voilà peut-être un début de réponse. Cet exemple montre aussi que le public ciblé détermine le meilleur choix pour la technologie de base. Le développeur lui-même devrait avoir le recul nécessaire pour prendre la décision la plus opportune.

Préparer minutieusement le terrain

Dans le même ordre d'idées, il faut créer un code lisible et bien documenté dès le début. Dans un environnement universitaire, certains projets logiciels rassemblent de nombreux contributeurs temporaires. Les changements dans les plannings et les projets des étudiants peuvent nuire à la qualité du code. Je me souviens d'un petit projet de logiciel à l'Université technique de Dresde qui avait été très bien maintenu par un assistant étudiant. Après son départ, on a constaté que le code était minutieusement documenté… en turc. Un chercheur ne peut être programmeur qu'à temps partiel. Une discipline particulière est donc nécessaire pour mettre en œuvre le travail supplémentaire indispensable à l'élaboration d'un code accessible. En retour, il y aura de bien meilleures chances par la suite d'avoir de bons rapports de bogues, des correctifs utiles ou même des développeurs extérieurs.

Semer les graines des communautés

Les développeurs *open source* inexpérimentés considèrent souvent comme un grand moment la publication ouverte de leur code. En réalité, personne d'autre qu'eux ne la remarquera. Pour attirer aussi bien des utilisateurs que des contributeurs, il faut faire passer le mot. La communication publique d'un vrai projet devrait au moins inclure des annonces à chaque nouvelle version. Les listes de diffusion sont probablement le meilleur canal pour cela. Il faut un certain talent social pour trouver le juste équilibre entre le spam indésirable et la litote timide. Si un projet est

motivé par la conviction sincère qu'il aidera les utilisateurs à résoudre de vrais problèmes, ce devrait être facile de lui faire une publicité convenable. Les utilisateurs feront vite la différence entre publicité éhontée et information utile. Bien évidemment, les annonces actives devront attendre que le projet soit finalisé. Cela ne concerne pas seulement le code, mais aussi la page d'accueil et la documentation pour une utilisation basique.

Au cours de sa vie, le projet devrait être mentionné dans tous les endroits *adéquats*, y compris des sites web – à commencer par votre page d'accueil ! –, des conférences, des papiers scientifiques, des discussions en ligne. On ne rendra jamais suffisamment grâce au pouvoir du simple lien qui conduira un futur contributeur important à sa première visite sur le site du projet. Les chercheurs ne doivent pas non plus oublier de publier leur logiciel en dehors de leur communauté universitaire proche. Les autres chercheurs sont rarement la meilleure base pour une communauté active.

Fournir des espaces pour grandir

Les responsables du projet se doivent de fournir des espaces de communication afin que la communauté puisse se développer. C'est un service banal mais souvent négligé. Sans liste de diffusion dédiée, toutes les demandes d'aide seront envoyées en message privé à la maintenance. Sans logiciel de suivi des problèmes[1], les rapports de bogues seront moins nombreux et moins utiles. Sans un wiki où tout un chacun pourra modifier la documentation utilisateur, le développeur est condamné à étendre et à réécrire la documentation en permanence. Si la version de développement du code source n'est pas accessible, alors les utilisateurs n'auront aucun moyen de tester la dernière version avant de se plaindre de problèmes. Si le dépôt de code est conçu pour être fermé, il ne sera alors pas possible d'accueillir des contributeurs externes. Un certain nombre de fournisseurs de service

1 *Bugtracker* (voir lexique).

proposent gratuitement un accès à toute cette infrastructure. Certaines formes d'interaction pourraient ne pas être souhaitables : par exemple, il y a des raisons pour que le cercle des développeurs reste fermé. Mais il ne serait pas raisonnable d'espérer le soutien d'une communauté sans même préparer un minimum d'espaces à son intention.

Encourager et contrôler la croissance

Les développeurs inexpérimentés sont souvent inquiets à l'idée que l'ouverture de listes de diffusion, de forums et de wikis pour les utilisateurs nécessitera une maintenance supplémentaire. C'est rarement le cas mais certaines activités de base sont bien entendu indispensables. Cela commence par *la mise en œuvre* d'une communication publique et son utilisation *rigoureuse*. Les utilisateurs ont besoin d'apprendre à poser des questions publiques, à consulter la documentation avant de poser des questions, et à rapporter les bogues à l'aide d'un logiciel de suivi des problèmes plutôt que par courriel. J'ai tendance à rejeter toutes les demandes d'aide privées ou à répondre sur des listes publiques. Cela permet par la même occasion de s'assurer que des solutions seront accessibles à de futurs utilisateurs qui feront une recherche sur le Web. Dans tous les cas, les utilisateurs doivent être remerciés explicitement pour toutes les formes de contributions : il faut beaucoup de personnes enthousiastes et bien intentionnées pour construire une communauté solide.

Quand on atteint un certain nombre d'utilisateurs, une aide mutuelle commence à se mettre en place entre eux. C'est toujours un moment magique pour un projet et c'est un signe évident qu'il est sur la bonne voie. Dans l'idéal, les responsables du projet devraient continuer d'apporter leur aide pour les questions délicates, mais à un moment donné certains utilisateurs vont faire preuve d'initiative dans les discussions. Il est important de les remercier (personnellement) et de les impliquer davantage dans le projet. À l'inverse, les évolutions malsaines doivent être stoppées

dès que possible, en particulier les comportements agressifs qui peuvent être un véritable danger pour le développement de la communauté. De même, l'enthousiasme le mieux intentionné n'est pas toujours productif et il faut parfois savoir dire non – gentiment mais clairement – pour éviter les dérapages possibles.

Le futur est ouvert

Construire une communauté initiale autour d'un projet contribue fortement à transformer un prototype de recherche en un logiciel *open source*. Si ça porte ses fruits, il existe de nombreuses options pour le développer en fonction des buts fixés par le responsable du projet et la communauté. Voici quelques indications :

- Persévérer dans l'expansion du projet et de sa communauté *open source*, augmenter le nombre de personnes ayant des droits de contributions « directs », en réduisant la dépendance à son origine universitaire. Par ce biais, vous impliquez plus fortement la communauté – notamment au travers d'évènements dédiés – et vous pourrez établir un soutien organisationnel ;
- Créer une entreprise commerciale pour exploiter le projet, basée, par exemple, sur une double licence ou un *business model* de consultant. Des outils ayant fait leurs preuves et une communauté active sont des atouts majeurs dès le lancement d'une entreprise, et peuvent être bénéfiques dans chacune de vos stratégies d'entreprise sans abandonner le produit original *open source* ;
- Se retirer du projet. Il y a de nombreuses raisons pour qu'on ne puisse plus maintenir de lien avec le projet. Le fait d'avoir établi une communauté saine et ouverte maximise les chances pour que le projet continue de voler de ses propres ailes. Dans tous les cas, il est plus correct de faire une coupure nette que d'abandonner silencieusement le

> projet, en le tuant par une activité en chute libre au point qu’on ne trouve plus personne pour le maintenir.

Le profil de la communauté sera différent selon qu’on opte pour telle ou telle stratégie de développement. Mais dans tous les cas, le rôle du chercheur évolue en fonction des objectifs du projet. Le scientifique initial et le programmeur pourront endosser la fonction de responsable ou de directeur technique. En ce sens, la différence principale entre un projet de logiciel *open source* d’importance et la recherche perpétuelle d’un prototype n’est pas tant la quantité de travail mais le type de travail nécessaire pour y arriver. Cela compte pour beaucoup dans sa réussite. Une fois qu’on l’a compris, il ne reste plus qu’à réaliser un super logiciel.

4. Préparez-vous pour le futur

L’évolution des équipes dans le logiciel libre et *open source*

Felipe Ortega

Felipe Ortega est chercheur et chef de projet à Libresoft, un groupe de recherche de l’Université Rey Juan Carlos en Espagne. Il développe de nouvelles méthodologies pour analyser les communautés collaboratives ouvertes (comme les projets de logiciels libres, Wikipédia et les réseaux sociaux). Il a mené des recherches approfondies sur le projet Wikipédia et sa communauté de contributeurs. Felipe participe activement à la recherche, la promotion et l’éducation/formation sur le logiciel libre, plus particulièrement dans le cadre du Master « Logiciel libre » de l’URJC[1]. *C’est un fervent défenseur de l’ouverture des ressources éducatives, du libre accès aux publications scientifiques et de l’ouverture des données scientifiques.*

Dans son célèbre essai La Cathédrale et le Bazar[2], Eric S. Raymond souligne l’une des plus importantes leçons que doit apprendre chaque développeur : « Un bon logiciel commence toujours avec un développeur qui va creuser un truc qui le démange ». Vous ne pouvez comprendre à quel point cette phrase est vraie qu’à condition d’avoir vous-même vécu la situation. En fait, la plupart des programmeurs de logiciels libres et *open source* (si ce n’est tous) ont sûrement éprouvé ce phénomène en

1 http://master.libresoft.es.
2 http://www.linux-france.org/article/these/cathedrale-bazar/cathedrale-bazar.html.

mettant les mains dans le cambouis sur un tout nouveau projet ou en rejoignant un projet en cours, impatients de participer à son amélioration. Cependant, bien des développeurs et autres participants dans les communautés libres et *open source* (rédacteurs de documentation, traducteurs, etc.) négligent généralement une autre leçon importante que souligne Raymond plus loin dans son essai : « Quand un programme ne vous intéresse plus, votre dernier devoir à son égard est de le confier à un successeur compétent ». C'est le thème central que je veux traiter ici. Vous devez penser à l'avenir de votre projet et aux nouveaux arrivants qui un jour prendront le relais et continueront de le faire avancer.

Le relais entre les générations

Tôt ou tard, de nombreux projets libres et *open source* devront faire face à un relais générationnel. Les anciens développeurs en charge de la maintenance du code et de ses améliorations finissent par quitter le projet et sa communauté pour des raisons diverses et variées. Il peut s'agir de problèmes personnels, d'un nouveau travail qui ne laisse pas assez de disponibilités, du démarrage d'un nouveau projet, ou du passage à un autre projet qui semble plus attirant… la liste peut être assez longue.

L'étude du relais générationnel (ou renouvellement des développeurs) dans les projets de logiciel libre et *open source* reste un domaine émergent qui nécessite davantage de recherches pour améliorer notre compréhension de ces situations. En dépit de cela, certains chercheurs ont déjà collecté des preuves objectives qui mettent en lumière certains processus. Pendant l'OSS 2006[1], mes collègues Jesús González-Barahona et Gregorio Robles présentèrent un travail intitulé « Le renouvellement des contributeurs dans les projets de logiciel libre ». Dans cette présentation, ils exposèrent une méthodologie pour identifier les développeurs les plus actifs – généralement connus comme les développeurs principaux – à différents moments, pendant toute la durée d'un projet

1 Conférence sur l'Open Source System – http://oss2006.org.

donné. Ils appliquèrent ensuite cette méthode à l’étude de 21 gros projets, en particulier GIMP[1], Mozilla[2] et Evolution[3]. En bref, ils ont découvert qu’on peut distinguer trois types de projets en fonction du taux de renouvellement des développeurs.

Les projets avec des dieux du code : ces projets reposent en grande partie sur le travail de leurs fondateurs et le relais générationnel est très faible, voire nul. GIMP se classe dans cette catégorie ;

Les projets avec de multiples générations de codeurs : des projets comme Mozilla montrent clairement un modèle de renouvellement des développeurs, avec de nouveaux groupes actifs qui prennent en main la gestion du développement et de la maintenance du code des mains mêmes du noyau des anciens contributeurs ;

Les projets composites : *Evolution* appartient à une troisième catégorie de projets ; il connaît un certain taux de renouvellement, mais celui-ci n’est toutefois pas aussi marqué que pour les projets de la catégorie précédente, parce qu’atténué par la rétention de certains des principaux contributeurs au cours de l’histoire du projet.

Cette classification nous amène à une question évidente : quel est le modèle le plus fréquemment rencontré dans les projets de logiciels libres et *open source* ? Pour tout dire, les résultats de l’analyse menée sur l’échantillon de 21 projets lors de ces travaux établissent clairement cette conclusion : ce sont les projets à multiples générations, ainsi que les projets composites qui sont les plus communément rencontrés dans l’écosystème des projets libres et *open source*. Seuls Gnumeric et Mono ont montré un modèle distinct avec une forte rétention d’anciens développeurs, ce qui indique que les personnes contribuant à ces projets auraient de plus fortes raisons de continuer leurs travaux sur le long terme.

1 Logiciel de création graphique, http://www.gimp.org.
2 https://www.mozilla.org/fr/about.
3 Logiciel de messagerie, http://projects.gnome.org/evolution.

Ce n’est pourtant pas le cas le plus courant. Au contraire, cette étude donne plus de légitimité au conseil suivant : nous devons préparer, à plus ou moins long terme, le transfert de notre rôle et de nos connaissances au sein du projet vers les futurs contributeurs qui rejoignent notre communauté.

Le fossé de connaissances

Toute personne qui fait face à un changement significatif dans sa vie doit s’adapter à de nouvelles conditions. Par exemple, quand vous quittez votre emploi pour un autre, vous vous préparez à une période pendant laquelle il vous faudra vous intégrer à un nouveau groupe de travail, dans un nouveau lieu. Heureusement, au bout d’un moment vous aurez pris vos marques dans ce nouvel emploi. Mais vous aurez quelquefois gardé de bons amis de votre ancien boulot, que vous reverrez après votre départ. En discutant avec vos anciens collègues, vous pourrez alors peut-être savoir ce qu’il s’est passé avec la personne recrutée pour vous remplacer. Cela ne se produit que rarement dans les projets *open source*.

Le revers du relais générationnel dans un projet libre peut apparaître sous une forme très concrète, à savoir un fossé de connaissances. Quand un ancien développeur quitte le projet, et particulièrement s’il avait une expérience approfondie dans cette communauté, il laisse derrière lui ses connaissances aussi bien concrètes qu’abstraites, qui ne sont pas forcément transmises aux nouveaux venus[1].

Un exemple évident est le code source. Comme dans toute production intellectuelle bien faite – du moins, c’est ce à quoi on pourrait s’attendre, non ? –, les développeurs laissent une marque personnelle chaque fois qu’ils écrivent du nouveau code. Parfois, vous avez l’impression d’avoir une dette éternelle envers le programmeur génial qui a écrit ce code élégant et soigné qui parle de lui-même et qui est facile à maintenir. D’autres fois, la situa-

1 [NdT] L’original de cette phrase est au féminin : *she*… *she*… *her*…

tion est inverse et vous bataillez pour comprendre un code très obscur sans un seul commentaire ni indice pour vous aider.

C’est ce que l’on a essayé de mesurer en 2009, dans une recherche présentée à l’HICSS 2009[1]. Le titre en est « Utiliser l’archéologie logicielle pour mesurer les pertes de connaissances provoquées par le départ d’un développeur ». Au cas où vous vous poseriez la question, cela n’a rien à voir avec des histoires de fouet, de trésors, de temples, et autres aventures palpitantes. Ce qui a été mesuré, entre autres choses, c’est le pourcentage de code orphelin laissé par les développeurs ayant quitté des projets libres et *open source*, et qu’aucun des développeurs actuels n’a encore repris. Pour cette étude, nous avons choisi quatre projets (Evolution, GIMP, Evince et Nautilus) pour tester la méthode de recherche. Et nous sommes arrivés à des résultats assez intéressants.

Evolution présentait une tendance plutôt inquiétante car le taux de code orphelin augmentait au cours du temps. En 2006, près de 80 % de l’ensemble des lignes de code avaient été abandonnées par les précédents développeurs et étaient restées intouchées par le reste de l’équipe. À l’opposé, GIMP affichait un modèle tout à fait différent, avec une volonté claire et soutenue par l’équipe de développement de réduire le nombre de lignes orphelines. Souvenons-nous au passage que GIMP avait déjà été qualifié de projet des dieux du code et bénéficiait donc d’une équipe de développement bien plus stable pour surmonter cette tâche harassante.

Cela signifie-t-il que les développeurs de GIMP avaient une bien meilleure expérience que ceux d’Evolution ? Honnêtement, on n’en sait rien. Néanmoins, on peut prévoir un risque évident : plus le taux de code orphelin est élevé, plus l’effort à fournir pour maintenir le projet est important. Que ce soit pour corriger un bogue, développer une nouvelle fonctionnalité ou en étendre une préexistante, il faut faire face à du code que l’on n’a jamais vu

1 Hawaii International Conference on System Sciences. Archives de la conférence : http://www.informatik.uni-trier.de/~ley/db/conf/hicss/hicss2009.html.

auparavant. Bien sûr les programmeurs d'exception existent, mais aussi extraordinaire que soit cette exception, les développeurs de GIMP ont un avantage certain ici, puisqu'ils ont quelqu'un dans l'équipe qui a une connaissance précise de la majorité du code à maintenir. De plus, ils travaillent à réduire la portion de code inconnu au cours du temps.

C'est comme à la maison

Ce qui est intéressant, c'est que certains projets parviennent à retenir les utilisateurs sur des périodes bien plus longues qu'on aurait pu s'y attendre. Là encore, nous pouvons trouver des preuves empiriques à l'appui de cette déclaration. Pendant l'OSS 2005, Michlmayr, Robles et González-Barahona présentèrent des résultats pertinents concernant cet aspect. Ils étudièrent la persistance de la participation des responsables de logiciels sur Debian en calculant ce qu'on appelle la demi-vie. C'est le temps nécessaire à une population donnée de développeurs principaux pour perdre la moitié de sa taille initiale. Le résultat fut que la demi-vie estimée des responsables Debian était approximativement de 7 ans et demi. En d'autres termes, l'étude ayant été menée sur une période de six ans et demi (entre juillet 1998 et décembre 2004), donc depuis Debian 2.0 jusqu'à Debian 3.1 (versions stables uniquement), plus de 50 % des responsables de Debian 2.0 contribuaient encore à Debian 3.1.

Debian a créé une sorte de procédure très formelle pour accepter de nouveaux codeurs logiciels (aussi connus sous le nom de développeurs Debian) qui inclut l'acceptation du Contrat social Debian et la démonstration d'une bonne connaissance de la Politique Debian. Par conséquent, on peut s'attendre à avoir des contributeurs très engagés. C'est en effet le cas, puisque les auteurs de l'étude ont constaté que les paquets délaissés par les anciens développeurs étaient généralement repris par d'autres développeurs de la communauté. C'est seulement dans le cas où le paquet n'était plus utile qu'il a été abandonné. Je pense que

nous pouvons tirer quelques conclusions de ces travaux de recherche :

1. Passez du temps à développer les principales lignes directrices de votre projet. Cela peut commencer par un seul et court document, qui détaille simplement des recommandations et des bonnes pratiques. Cela devrait évoluer à mesure que le projet grandit et permettre aux nouveaux arrivants tant de saisir rapidement les valeurs principales de votre équipe, que de comprendre les traits principaux de votre méthodologie.
2. Forcez-vous à suivre des standards de codage, des bonnes pratiques et un style élégant. Documentez votre code. Insérez des commentaires pour décrire les sections qui seraient particulièrement difficiles à comprendre. Ne pensez pas que c'est du temps perdu. En fait, vous faites preuve de pragmatisme en investissant du temps dans l'avenir de votre projet.
3. Dans la mesure du possible, lorsque le moment vient pour vous de quitter le projet, essayez d'avertir les autres de cette décision longtemps à l'avance. Assurez-vous qu'ils comprennent quelles parties essentielles du code nécessiteront un nouveau développeur pour le maintenir. Idéalement, si vous formez une communauté, préparez au moins une procédure simple afin d'automatiser la transition, et assurez-vous de n'oublier aucun point important avant que la personne ne quitte le projet (particulièrement si celle-ci est un développeur clé).
4. Gardez un œil sur la quantité de code orphelin. Si celle-ci augmente trop rapidement, ou si elle atteint une trop grande proportion de votre projet, cela indique clairement que vous allez avoir des problèmes dans peu de temps, en particulier si le nombre de rapports de bogues augmente ou si vous envisagez une nouvelle implémentation de votre code avec de fortes modifications.

5. Assurez-vous toujours de laisser assez d’astuces et de commentaires pour qu’à l’avenir un nouvel arrivant puisse s’approprier votre travail.

J’aurais voulu savoir que vous arriviez (avant de partir)

Je reconnais que ce n’est pas très facile de penser à ses successeurs lorsque vous programmez. La plupart du temps, vous ne vous rendez tout simplement pas compte que votre code pourrait à la fin être repris dans un autre projet, réutilisé par d’autres personnes ou que vous pourriez éventuellement être remplacé par quelqu’un d’autre, qui continuera votre travail après vous. Cependant, le plus remarquable atout des logiciels libres et *open source* est précisément celui-là : le code sera réutilisé, adapté, intégré ou exporté par d’autres. La maintenance est une composante essentielle de l’ingénierie logicielle. Mais cela devient primordial dans le cas des logiciels libres et *open source*. Ce n’est pas seulement une question de code source. Cela concerne aussi les relations humaines et la netiquette. C’est quelque chose qui va au-delà du bon goût. *Quod severis metes* (« On récolte ce que l’on a semé »). Souvenez-vous-en. La prochaine fois, vous pourriez être le nouveau venu qui viendra combler le vide de connaissance laissé par un ancien développeur.

Guider et recruter

5. Vous finirez par savoir tout ce qu'ils ont oublié

Leslie Hawthorn

Gestionnaire de communautés internationalement reconnue, conférencière et auteure, Leslie Hawthorn a plus de 10 ans d'expérience dans la gestion de projets high-tech, le marketing et les relations publiques. Elle a récemment rejoint AppFog[1] *en tant que responsable de la communauté, où elle est chargée du recrutement de développeurs. Auparavant, elle a travaillé comme responsable de communication au laboratoire* open source *de l'Université d'Oregon et comme responsable de programme au sein de l'équipe* open source *de Google, où elle a géré le* Google Summer of Code[2]*, créé le concours que l'on connaît maintenant sous le nom Google Code-in et lancé le blog de développement* open source *de la société.*

> « La documentation la plus importante pour les nouveaux utilisateurs concerne les bases : comment installer rapidement le logiciel, avoir une vue d'ensemble de son fonctionnement et peut-être quelques guides pour les tâches courantes. Or, voilà précisément ce que les auteurs de la documentation ne connaissent que trop parfaitement. Si parfaitement, qu'il peut être difficile pour eux de voir les choses du point de vue du lecteur et d'énumérer laborieusement les étapes qui (aux yeux des auteurs)

1 http://www.appfog.com.
2 http://fr.wikipedia.org/wiki/Google_Summer_of_Code.

> semblent évidentes ou inutiles à mentionner. » Karl Fogel, *Produire du logiciel libre*[1].

Quand pour la première fois vous commencez à travailler sur un projet de logiciel libre et *open source*, la courbe d'apprentissage est raide et le chemin difficile. Vous risquez de vous retrouver abonné à des listes de diffusion ou dans des salons de discussion avec toutes sortes de gens renommés, comme le créateur de votre langage de programmation favori ou le responsable de votre logiciel préféré, et vous vous demanderez si vous serez un jour suffisamment qualifié pour contribuer efficacement. Ce dont vous n'aurez pas forcément conscience, c'est à quel point ces gens sages ont oublié le long chemin qui les a menés au succès.

Prenons une analogie simple : dans un projet *open source*, le processus d'apprentissage, comme utilisateur ou comme développeur, c'est un peu comme apprendre à faire du vélo. Pour les cyclistes expérimentés, « c'est aussi facile que de monter à vélo ». Vous avez probablement fait du vélo quelquefois et vous comprenez son architecture : une selle, des roues, des pédales et un guidon. Pourtant, vous montez en selle, concentré sur votre avancée et soudainement vous découvrez que ce n'est pas aussi simple que ce que vous pensiez : à quelle hauteur faut-il régler votre selle ? Quel équipement vous faut-il quand vous grimpez une colline ? Quand vous en descendez une ? D'ailleurs, avez-vous vraiment besoin de ce casque ? (un conseil : oui, absolument).

Lorsque vous vous mettez au vélo, vous ne savez même pas quelles questions poser et vous ne les trouverez que dans vos genoux endoloris, des points de côté et des courbatures dans le dos. Même dans ce cas, vos questions ne correspondront pas toujours aux réponses dont vous avez besoin ; quelqu'un pourrait s'aviser de vous dire d'abaisser la selle quand vous lui dites que vos genoux font mal, mais d'autres peuvent tout aussi bien supposer que cela étant nouveau pour vous, vous finirez bien par le découvrir par vous-même. Ils ont oublié qu'il faut se battre avec

1 http://framabook.org/8-produire-du-logiciel-libre.

les changements de vitesse, se rendre compte qu'on n'a pas les bons éclairages ni les réflecteurs adéquats, apprendre comment tourner à gauche en levant la bonne main, parce qu'ils font du vélo depuis si longtemps que tous ces gestes sont pour eux comme une seconde nature.

C'est le même scénario lorsque vous débutez dans le monde des logiciels libres et *open source*. Lorsque vous compilez un paquet pour la première fois, vous allez inévitablement arriver à un obscur message d'erreur ou un autre genre d'échec. Et lorsque vous demanderez de l'aide, une bonne âme vous dira sans doute : « c'est facile, il suffit de faire make -toto -titi -tata ». Sauf que pour vous, ce n'est pas facile. Il n'y aura probablement pas de documentation pour toto, titi ne fera pas ce qu'il est supposé faire et qu'est-ce que ce truc tata avec ses huit homonymes sur Wikipédia ? Évidemment, vous ne voulez pas être un boulet, mais vous allez avoir besoin d'aide pour réussir vraiment à faire quelque chose.

Vous allez peut-être persister à reprendre les mêmes étapes, rencontrer les mêmes échecs, et la frustration ira grandissant. Peut-être que vous allez vous lever pour prendre un café en pensant que vous reviendrez sur le problème plus tard. Ce qu'aucun de nous dans le monde des logiciels libres et *open source* ne voudrait voir se produire, c'est précisément ce qui se passe pour beaucoup : boire cette tasse de café est infiniment meilleur que de se sentir ignorant et intimidé, et vous n'allez pas plus avant dans votre découverte du Libre.

Prenez conscience dès maintenant que vous finirez par connaître ces choses que les experts autour de vous ont oubliées ou qu'ils ne communiquent pas car ces étapes sont évidentes pour eux. Toute personne plus expérimentée que vous est passée par les mêmes affres que vous en ce moment pour apprendre à faire ce que vous vous efforcez de faire. Voici quelques conseils pour rendre votre parcours plus facile.

N’attendez pas trop longtemps avant de demander de l’aide. Personne ne veut être un boulet et personne n’aime avoir l’air perdu. Cela dit, si vous n’arrivez pas à résoudre votre problème après avoir passé un quart d’heure à essayer, il est temps de demander de l’aide. Vérifiez la documentation sur le site web du projet afin d’utiliser le canal IRC, le forum ou la liste de diffusion dédiés pour demander de l’aide. De nombreux projets mettent en ligne des canaux d’aide spécialement pour les débutants, gardez donc un œil sur des mots tels que *mentor*, *débutant* et *mise en route*.

Parlez de votre processus (de réflexion). Il ne s’agit pas seulement de poser des questions, mais de savoir quelles sont les bonnes questions à poser. Au début, vous ne saurez pas forcément quelles sont ces bonnes questions. Donc quand vous demanderez de l’aide, détaillez ce que vous essayez de faire, les étapes par lesquelles vous êtes passé, et les problèmes que vous avez rencontrés. Signalez aux futurs mentors du canal IRC ou de la liste de diffusion que vous avez consulté le manuel en incluant des liens vers la documentation que vous avez lue sur le sujet. Si vous n’avez trouvé aucune documentation, le signaler poliment peut aider.

Apprenez à connaître votre propre valeur. En tant que nouveau contributeur dans un projet, vous êtes un atout précieux. Non pas pour vos connaissances, mais pour votre ignorance. Lorsque vous commencez à travailler sur des logiciels libres et *open source*, rien n’est assez évident à vos yeux et tout mérite donc d’être expliqué. Prenez des notes à propos des problèmes que vous avez rencontrés et sur la façon dont ils ont été résolus. Puis utilisez ces notes pour mettre à jour la documentation du projet, travailler avec la communauté à des captures vidéo ou autres documents de formation pour les cas les plus épineux.

Quand vous rencontrez un problème vraiment frustrant, comprenez que vous êtes en position idéale pour faire en sorte que le prochain qui tombera dessus ne rencontre pas les mêmes difficultés.

6. Université et communauté

Kevin Ottens

Kevin Ottens est un contributeur passionné et de longue date au sein de la communauté KDE[1]*. Il a largement contribué à la plateforme KDE, en particulier à la conception des API et frameworks. Diplômé en 2007, il est titulaire d'un doctorat en informatique qui l'a amené à travailler, en particulier, sur l'ingénierie des* ontologies[2] *et les systèmes multi-agents. Le travail de Kevin au KDAB inclut le développement de projets de recherche autour des technologies KDE. Il vit toujours à Toulouse, où il est enseignant à mi-temps dans son université d'origine.*

Introduction

Les communautés du Libre sont principalement animées par l'effort de bénévoles. De plus, la plupart des personnes qui s'impliquent dans ces communautés le font lors de leur cursus universitaire. C'est la période idéale pour s'engager dans de telles aventures : on est jeune, plein d'énergie, curieux, et l'on veut probablement façonner le monde à son image. Voilà tous les ingrédients d'un bon travail bénévole.

Mais, en même temps, être étudiant ne laisse pas forcément beaucoup de temps pour s'engager dans une communauté du

1 http://www.kde.org.
2 https://fr.wikipedia.org/wiki/Ontologie_(informatique).

Libre. En effet, la plupart de ces communautés sont assez vastes et les contacter peut faire peur.

Cela soulève évidemment une question dérangeante : si les communautés du Libre ne réussissent pas à attirer la nouvelle génération de contributeurs talentueux, est-ce parce qu'elles ne cherchent pas activement à étendre leur activité dans les universités ? Cette question pertinente, nous avons essayé d'y répondre dans le contexte d'une communauté qui produit des logiciels, à savoir KDE. Dans cet article, nous nous concentrerons sur les aspects auxquels nous n'avions pas initialement pensé mais qu'il nous a fallu aborder en cherchant à répondre à cette question.

Construire un partenariat avec une université locale

Tout commence réellement par le contact avec les étudiants eux-mêmes. Et pour ça, rien de mieux que de se rendre directement dans leur université et leur montrer à quel point les communautés du Libre peuvent être accueillantes. À cet effet, nous avons construit un partenariat avec l'université Paul Sabatier de Toulouse, plus précisément, avec l'un de ses cursus – nommé IUP ISI[1] – axé sur le développement logiciel.

L'IUP ISI est très orienté sur les connaissances « pratiques » et propose à ce titre un programme pré-établi pour les projets étudiants. Un point particulièrement intéressant de ce programme est le fait que les étudiants travaillent en équipe « inter-promotions ». Des étudiants de troisième et quatrième années, généralement en équipes de 7 à 10, apprennent à collaborer autour d'un objectif commun.

La première année de notre expérience, nous nous sommes raccordés à ce programme en proposant de nouveaux sujets pour les projets, et en nous concentrant sur des logiciels développés au sein de la communauté KDE. Henri Massié, directeur du cursus, s'est montré très réceptif à cette idée et nous a laissés mettre cette

1 [NdT] Ingénierie des Systèmes Informatiques.

expérience en place. Pour cette première année, nous nous sommes vu attribuer deux créneaux horaires pour les « projets KDE ».

Afin de créer rapidement un climat de confiance, nous avons alors décidé d'offrir quelques garanties concernant le travail des étudiants.

Pour aider les professeurs à avoir confiance dans les sujets abordés, les projets sélectionnés étaient très proches des sujets enseignés à l'IUP ISI (c'est pourquoi, cette année-là, nous avons ciblé un outil de modélisation UML et un outil de gestion de projet).

Pour donner un maximum de visibilité aux professeurs, nous avons mis à leur disposition un serveur accessible à distance à des fins de test, sur lequel étaient régulièrement compilés les projets des étudiants.

Pour faciliter la participation des étudiants à la communauté, les responsables des projets étaient désignés pour jouer le rôle du « client », soumettre leurs exigences aux étudiants et les aider à trouver leur chemin dans le dédale de la communauté.

Enfin, **pour mettre le pied à l'étrier aux étudiants**, nous leur avons donné un petit cours sur « Comment développer avec Qt et les autres frameworks produits par KDE ».

Au moment où j'écris de ces pages, cela fait cinq ans que nous menons de tels projets. De petits ajustements dans l'organisation ont été apportés ici et là, mais la plupart des idées sous-jacentes sont restées les mêmes. La majorité des changements ont été le résultat d'un intérêt grandissant de la communauté, désireuse d'établir un partenariat avec les étudiants, et d'une plus grande liberté pour nous quant aux sujets que nous pouvions couvrir dans nos projets.

De plus, tout au long de ces années, le directeur nous a accordé une aide et des encouragements constants de manière effective, il nous a attribué plus de créneaux pour les projets de la commu-

nauté du Libre. Cela prouve que notre stratégie d'intégration était juste : établir la confiance dès le début est la clé d'un partenariat entre la communauté du Libre et l'Université.

Comprendre que l'enseignement est un processus interactif

Au cours de ces années passées à tisser des liens entre la communauté KDE et la filière IUP ISI, nous avons fini par nous retrouver en situation d'enseigner à des étudiants afin de les assister dans des tâches liées à leurs projets. Quand vous n'avez jamais enseigné à une classe pleine d'étudiants, vous avez sans doute encore en tête une image de vous-même, il y a quelques années, assis dans une classe. En effet, la plupart des enseignants ont un jour été étudiants… Parfois même, vous n'étiez pas du genre à vous montrer très discipliné ni attentif. Sans doute aviez-vous alors l'impression d'être submergé : l'enseignant entrait dans la salle, faisait face aux étudiants, et déversait sur vous ses connaissances.

Ce stéréotype est ce que la plupart des personnes ont retenu de leurs années d'études et la première fois qu'ils se retrouvent en situation d'enseigner, ils veulent le reproduire et arriver avec un savoir à transmettre.

La bonne nouvelle, c'est que rien n'est plus éloigné de la vérité que ce stéréotype. La mauvaise nouvelle, c'est que si vous essayez de le reproduire, vous allez très probablement faire fuir vos étudiants et ne ferez face qu'à un manque de motivation pour participer à la communauté. L'image que vous donnez de vous est la toute première chose qu'ils retiendront de la communauté : la première fois que vous entrez dans la salle de classe, *vous êtes*, pour eux, la communauté.

Pour éviter de tomber dans le piège de ce stéréotype, il vous faut prendre un peu de recul et prendre conscience de ce que signifie réellement enseigner. Ce n'est pas un processus à sens

unique où l’on livre la connaissance aux étudiants. Nous sommes arrivés à la conclusion que c’est plutôt un processus à double sens : vous êtes amené à créer une relation symbiotique avec vos étudiants. Les étudiants comme les enseignants doivent tous sortir de la salle de classe avec de nouvelles connaissances. Il vous faut transmettre votre expertise, bien sûr, mais afin de le faire efficacement, vous devez en permanence vous adapter au cadre de référence de vos étudiants. C’est un travail qui rend très humble.

Cette prise de conscience génère pas mal de changements dans la manière d’entreprendre votre enseignement.

- Vous allez devoir comprendre la culture de vos étudiants. Ils ont probablement des expériences assez différentes des vôtres et vous allez devoir adapter votre discours à eux ; par exemple, les étudiants que nous avons formés font tous partie de la fameuse « génération Y » qui, en matière de leadership, loyauté et confiance, présente des caractéristiques assez différentes de la génération précédente.
- Du fait d'avoir adapté votre discours à leur culture, vous serez amené à réévaluer votre propre expertise. Vous aborderez vos propres connaissances selon un angle très différent de celui dont vous avez l’habitude et cela vous mènera inévitablement à des découvertes dans des domaines que vous pensiez maîtriser.
- Enfin, vous allez devoir vous forger des compétences en présentation ; l’enseignement consiste vraiment à sortir de votre zone de confort afin de présenter vos propres connaissances tout en les gardant intéressantes et divertissantes pour votre audience. Cela fera de vous un meilleur présentateur.

Vous deviendrez ainsi un meilleur enseignant. De plus, vous remplirez mieux vos objectifs : des étudiants bien formés, dont certains s’engageront dans la communauté du Libre.

Conclusion

En fin de compte, pourquoi feriez-vous tous ces efforts pour établir une relation de confiance avec une université et sortir de votre zone de confort en améliorant votre manière d'enseigner ? Eh bien, cela se résume vraiment à la question initiale à laquelle nous avons tenté de répondre :

> *Si les communautés du Libre ne réussissent pas à attirer de nouveaux contributeurs venus des universités, est-ce simplement dû à leur inaction ?*

D'après notre expérience, la réponse est oui. Au cours de ces cinq années passées à bâtir un partenariat avec l'IUP ISI, nous avons attiré en moyenne deux étudiants par an. Certains d'entre eux nous ont quittés après quelque temps, mais quelques-uns sont devenus des contributeurs très actifs. Les autres gardent encore une certaine nostalgie de cette période de leur vie et continuent de nous soutenir même s'ils ne contribuent pas directement. En ce moment même, nous avons une équipe locale KDE qui a réussi à organiser efficacement une conférence de deux jours pour notre dernière « release party »[1].

Parmi ces anciens étudiants, pas un seul ne se serait impliqué dans le projet KDE sans ces projets universitaires. Nous serions passés complètement à côté de ces talents. Fort heureusement, nous n'avons pas été inactifs.

1 [NdT] soirée de lancement.

7. Laisser le champ libre

Lydia Pintscher

Lydia Pintscher est une femme naturellement douée pour apprivoiser les gens, les geeks et les chatons. Entre autres, elle a la responsabilité des programmes de mentors de KDE (Google Summer of Code, Google Code--in, Season of KDE), et fait partie des membres fondateurs des groupes de travail de la communauté KDE et des membres du conseil de KDE e.V[1].

Le logiciel libre a un ennemi. Ce n'est pas celui auquel pensent la plupart des personnes sur Internet. Non, l'ennemi, c'est le manque de participation active.

Chaque jour des milliers de personnes se mettent à chercher ce qui pourrait bien donner un sens à leur vie et se demandent comment réaliser quelque chose qui compte vraiment. Tous les jours des milliers de lignes de code pour des logiciels libres sont en attente d'écriture et de débogage, des programmes attendent d'être mis en avant et traduits, des créations artistiques attendent de voir le jour et ainsi de suite.

Malheureusement, il arrive bien trop souvent que la connexion ne s'établisse pas entre tous ces individus et tous ces projets. Il y a plusieurs raisons à cela. Au départ les gens ne connaissaient rien au logiciel libre, à tous ses avantages et à ses finalités. Mais on y

1 http://ev.kde.org.

vient peu à peu. Les gens se mettent à utiliser et peut-être même à comprendre le logiciel libre à grande échelle. Les projets de logiciels libres vivent de la conversion de certains de leurs utilisateurs en contributeurs actifs. Et c'est là que les problèmes sérieux commencent.

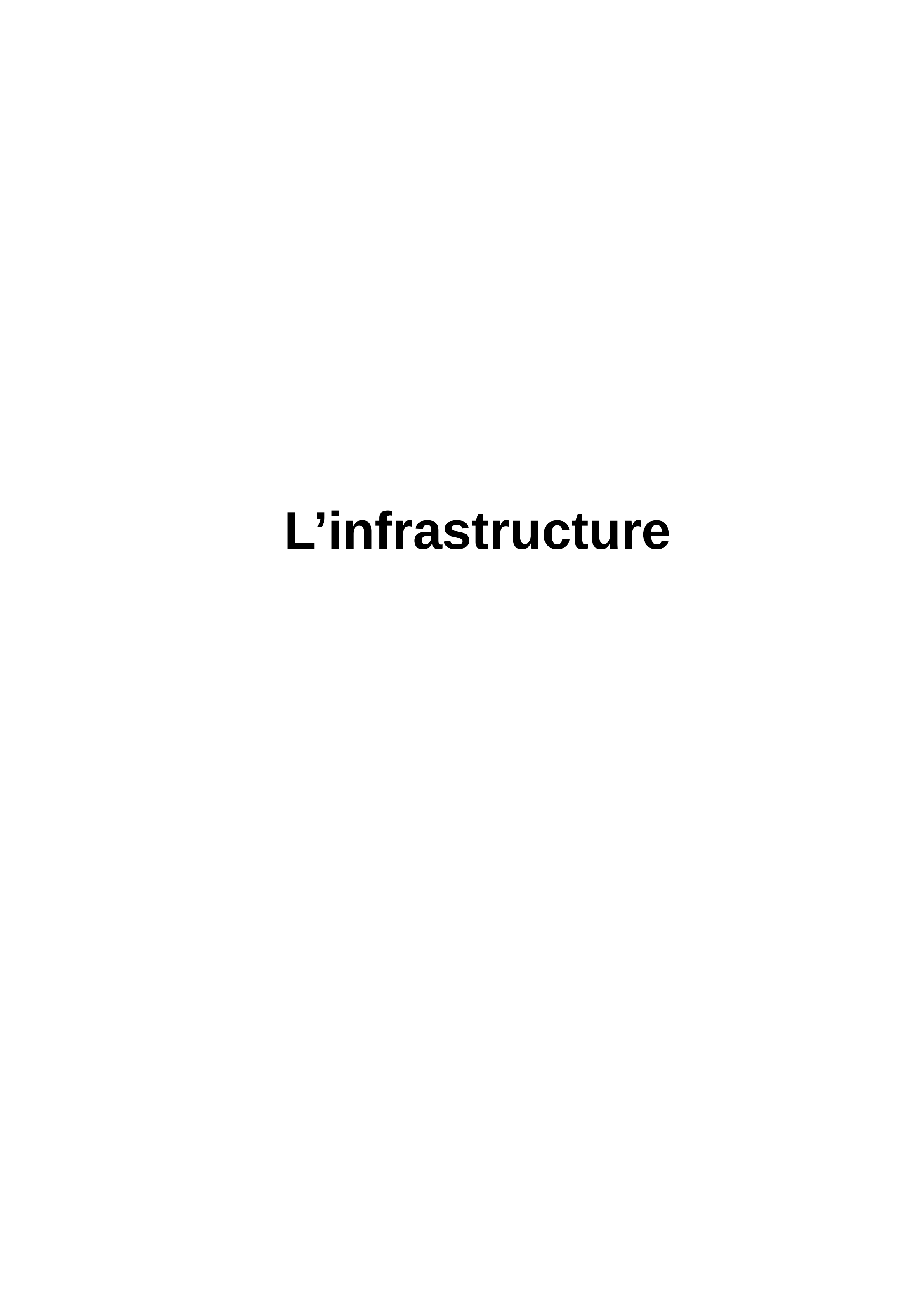
J'ai accompagné des centaines d'étudiants dans des programmes de tutorat et j'ai participé de diverses façons à des actions de sensibilisation aux projets de logiciels libres. J'ai travaillé avec des gens enthousiastes dont la vie a pris un tour bien meilleur grâce à leurs contributions aux logiciels libres. Mais il y a une rengaine que je ne cesse d'entendre et qui me brise le cœur parce que je sais maintenant à côté de quels talents nous passons à cause de ça : on ne s'est pas senti autorisé à faire quelque chose d'extraordinaire. Un collègue tuteur du *Google Summer of Code* a dit cette phrase qui résume au mieux ce sentiment : « L'impression générale est rarement celle-ci : pour la plupart des contributeurs dans l'*open source*, ce n'est pas tant qu'ils n'ont pas eu la permission de travailler sur des trucs, mais simplement qu'ils n'ont pas été là assez vite au bon moment ». Les contributeurs potentiels pensent souvent qu'ils ne sont pas autorisés à contribuer. Les raisons en sont nombreuses et reposent toutes sur des malentendus. Voici les préjugés les plus courants d'après mon expérience.

> « Je ne sais pas coder. Pas moyen pour moi de contribuer. »
>
> « Je ne suis pas vraiment bon pour ça. On n'a pas besoin de mon aide. »
>
> « Je serais seulement un boulet. Ils ont des choses plus importantes à gérer. »
>
> « On n'a pas besoin de moi. Ils doivent avoir déjà assez de gens bien plus brillants que moi. »

Ces préjugés sont presque toujours sans fondement et j'aurais aimé savoir il y a bien longtemps qu'ils sont si répandus. J'aurais dès le début abordé très différemment mon travail de sensibilisation.

Le plus simple, pour sortir quelqu'un de cette situation, c'est de l'inviter en personne. « Cet atelier que nous proposons ? – Oui, bien sûr, tu devrais venir. » « Ce bogue remonté dans le traqueur de bogues ? – Je suis sûre que tu es la personne idéale pour essayer de le corriger. » « Ce communiqué de presse qu'il faut préparer ? – Ce serait super si tu pouvais le relire et t'assurer qu'il est bon. » Et si ce n'est pas possible, assurez-vous que votre matériel de promotion – vous en avez bien un peu, non ? – précise clairement quel genre de personnes vous recherchez et ce que vous estimez être les prérequis.

Assurez-vous surtout de toucher les personnes en dehors du cercle des contributeurs habituels, car la barrière est encore plus haute pour eux. À moins de dépasser cette limite, vous ne recruterez que vous-même – c'est-à-dire que vous aurez davantage de contributeurs semblables à ceux que vous avez déjà. Les personnes semblables à celles déjà présentes sont géniales, mais pensez à toutes les autres personnes extraordinaires à côté desquelles vous passez et qui pourraient apporter de nouvelles idées et de nouvelles compétences à votre projet.



L’infrastructure

8. Aimer l’inconnu

Jeff Mitchell

Jeff Mitchell passe ses journées de travail à s’activer sur tout ce qui touche aux ordinateurs et aux réseaux et son temps libre à barboter dans toutes sortes de projets de logiciels libres et open source. *Ce qu’il préfère c’est la convergence des deux. Après avoir travaillé en tant qu’administrateur systèmes professionnel de 1999 à 2005, il maintient son niveau de compétences en les mettant bénévolement au service de projets libres en divers lieux. Ces temps-ci, son activité pour le Libre est dédiée à l’administration systèmes pour* KDE[1] *et c’est l’un des développeurs principaux du lecteur* Tomahawk[2]. *Jeff vit actuellement à Boston, aux États-Unis.*

Récemment, à mon travail, j’ai fait partie d’une équipe qui faisait passer les entretiens d’embauche pour un poste d’administrateur systèmes. Après avoir parcouru quelques dizaines de curriculum vitae, nous avons finalement convoqué notre premier candidat. Celui-ci – appelons-le John – avait aussi bien l’expérience de petites structures, style laboratoire informatique, que de plus vastes opérations dans des centres de données. À première vue, les choses se présentaient bien, si ce n’est qu’il avait eu cette réponse bizarre à quelques-unes de nos questions : « je suis administrateur systèmes ». Le sens de cette phrase n’a

1 http://www.kde.org.
2 http://www.tomahawk-player.org.

pas été immédiatement clair pour nous, jusqu'à ce que l'échange suivant ait lieu :

> Moi — Donc, vous avez dit que vous n'avez pas d'expérience avec Cisco IOS, mais qu'en est-il des réseaux en général ?
>
> John — Eh bien, je suis administrateur systèmes.
>
> Moi — Oui, mais que diriez-vous sur les concepts de réseau ? Les protocoles de routage comme BGP ou OSPF, les VLANs, les ponts réseau…
>
> John, exaspéré — Je suis *administrateur systèmes*.

C'est à ce moment-là que nous avons compris ce qu'il voulait dire. John ne nous disait pas qu'il connaissait toutes ces choses que nous lui demandions puisqu'il était administrateur systèmes ; il nous expliquait que *parce qu*'il était administrateur systèmes, il n'en savait rien. John était administrateur *systèmes* et cela signifiait pour lui que ces tâches étaient celles d'un administrateur réseau. Sans surprise, John n'a pas obtenu le poste.

Dans bien des projets *open source*, la spécialisation est une malédiction et non une bénédiction. Qu'un projet relève d'une catégorie ou l'autre dépend souvent de la taille de l'équipe de développement ; la spécialisation à l'extrême peut entraîner de graves perturbations dans un projet en cas de départ d'un développeur, que ce soit en bons ou mauvais termes, qu'on le regrette ou non. Il en va de même pour les administrateurs systèmes de projets *open source*, bien que la pénurie générale de ces derniers semble autoriser aux projets une marge de tolérance parfois dangereuse.

L'exemple le plus énorme qui me vienne à l'esprit impliquait un projet spécifique dont le site de documentation (y compris toute celle de l'installation et de la configuration) était indisponible depuis plus d'un mois. La raison ? Le serveur était en panne et la seule personne qui en avait l'accès naviguait sur un « bateau

pirate » avec les membres du parti pirate suédois. C'est une histoire vraie.

Cependant, tous les points de défaillance ne sont pas dus à l'absence des administrateurs systèmes ; certains sont artificiels. Sur un gros projet, les décisions des droits d'accès à l'administration systèmes étaient assumées par un seul administrateur. Il ne s'était pas seulement réservé certains droits d'accès uniquement pour lui-même (vous l'aurez deviné : oui, il a disparu pendant un certain temps, et oui, cela a causé des problèmes) ; il avait aussi décidé de la façon dont les droits d'accès devaient être accordés, en fonction de la confiance qu'il portait personnellement au candidat. La « confiance », dans ce cas, se fondait sur une seule chose : non pas le nombre de membres de la communauté qui se portait garant pour cette personne, ni depuis combien de temps cette personne était un contributeur actif et fiable au sein du projet, ni même depuis combien de temps il connaissait lui-même cette personne dans le cadre de ce projet. Au lieu de cela, elle reposait sur la façon dont il connaissait personnellement quelqu'un, ce par quoi il entendait la façon dont il connaissait cet individu *en personne*. Vous imaginez bien à quel point cela est adapté à une équipe d'administrateurs systèmes disséminés sur toute la planète…

Bien sûr, cet exemple ne fait qu'illustrer la grande difficulté pour un administrateur systèmes *open source* de trouver le juste milieu entre sécurité et capacité. Les grandes entreprises peuvent se permettre d'avoir du personnel redondant, et ce, même si le travail se répartit selon différentes responsabilités ou domaines de sécurité. La redondance est importante. Mais qu'en est-il si la seule possibilité d'avoir une redondance pour l'administrateur systèmes est de prendre la première personne se présentant au hasard sur votre canal IRC ou une personne quelconque proposant son aide ? Comment pouvez-vous raisonnablement avoir confiance en cette personne, ses capacités et sa motivation ? Malheureusement, seuls les contributeurs principaux du projet ou

une petite partie d'entre eux peuvent savoir quand la bonne personne se présente en utilisant le même modèle de toile de confiance[1] qui sous-tend une grande partie du reste du monde *open source*. L'univers des projets *open source*, leurs besoins et les personnes qui veulent contribuer à un projet particulier forment une extraordinaire diversité ; par conséquent, la dynamique humaine, la confiance, l'intuition et la manière d'appliquer ces concepts à un projet *open source* sont de vastes sujets, bien au-delà de la thématique de ce court article.

Une chose importante a cependant facilité la découverte de cette ligne d'équilibre sécurité/capacité : l'essor des systèmes de gestion de versions distribués, ou DVCS[2]. Auparavant, les contrôles d'accès étaient primordiaux car le cœur de tout projet *open source* – son code source – était centralisé. Je me rends bien compte que beaucoup doivent penser : « Jeff, tu devrais pourtant le savoir, le cœur d'un projet, c'est sa communauté, pas son code ! ». Ma réponse est simple : les membres de la communauté vont et viennent, mais, si quelqu'un fait accidentellement un `rm -rf` sur tout l'arbre du système de gestion de versions de votre projet et que vous manquez de sauvegardes, combien de ces membres de la communauté vont continuer à s'investir dans le projet et aider à tout recommencer à zéro ? (Mes propos se basent sur une histoire vraie dans laquelle un membre de la communauté saoul qui s'énervait à déboguer un bout de code, lança un `rm -rf` sur toute sa contribution, avec l'intention de supprimer tout le code du projet. Par chance, il n'était pas administrateur systèmes et n'avait donc pas accès au dépôt central, et il était trop saoul pour se rappeler qu'il travaillait seulement sur une copie du projet.)

Le code du projet est son cœur ; les membres de sa communauté en sont l'énergie vitale. Privé de l'un ou de l'autre, vous aurez du mal à garder un projet vivant. Avec un logiciel de

1 http://fr.wikipedia.org/wiki/Toile_de_confiance.

2 [NdT] Distributed Version Control System, système de gestion de version distribué.

gestion de version[1] centralisé, si vous n'avez pas eu la présence d'esprit de mettre en place un système de sauvegarde régulier, vous pourrez, avec de la chance, ré-assembler l'arborescence complète du code source à partir des différents éléments contribués qu'auront gardés les autres personnes. Mais pour la majorité des projets, l'historique du code est aussi important que le code lui-même et cela, vous l'aurez tout de même entièrement perdu.

Ce n'est plus le cas désormais. Quand tous les clones locaux ont tout l'historique du projet et que des sauvegardes de secours peuvent être effectuées chaque nuit, en lançant une tâche planifiée aussi simple que `git pull`, les dépôts centralisés ne sont plus que des outils de coordination. Cela en diminue l'importance de quelques degrés. Le projet doit toujours être protégé contre les menaces aussi bien internes qu'externes : les systèmes non corrigés sont toujours vulnérables à des exploits bien connus. Un administrateur systèmes malveillant peut tout mettre sens dessus dessous, un système d'authentification déficient peut permettre l'entrée de codes malveillants dans la base, et un `rm -rf` accidentel sur le dépôt central peut toujours coûter cher en temps de développement. Mais ces défis peuvent être surmontés, et à l'ère des serveurs privés virtuels (VPS) abordables et des centres d'hébergement de données, les absences des administrateurs systèmes peuvent également être compensées. (Il vaut mieux cependant s'assurer d'avoir un accès redondant au DNS ! Oh et mettez aussi vos sites internet sur un dépôt vérifié et certifié [DVCS] , et faites des branches pour les modifications locales. Vous me remercierez plus tard.) Les DVCS permettent la redondance du cœur de votre projet pour trois fois rien, ce qui est une bonne façon d'aider les administrateurs systèmes à dormir la nuit et nous donne l'impression d'être un peu comme des maîtres du temps. Cela veut aussi dire que si vous n'êtes pas sur un DVCS, arrêtez de lire immédiatement et passez sur l'un d'eux. Ce n'est pas qu'une question d'espace de travail et d'outils. Si vous vous

1 [NdT] VCS pour version control system.

souciez de la sécurité de votre code et de votre projet, vous migrerez.

La redondance du code source est une nécessité, et en général plus vous avez de redondances, plus vos systèmes sont robustes. Il semble aussi évident que vous voulez une redondance de vos administrateurs systèmes ; ce qui vous semblera peut-être moins évident, c'est que l'importance de la redondance ne se joue pas tant en termes de personnes qu'en termes de niveau des compétences. John, l'administrateur systèmes, a travaillé dans des centres de stockage des données et au sein d'entreprises qui avaient des systèmes d'administration redondants mais des niveaux de compétences rigides, définis. Cela fonctionne dans de grandes entreprises qui peuvent payer pour embaucher de nouveaux administrateurs systèmes avec des compétences à la carte. Mais la plupart des projets *open source* n'ont pas ce luxe. Vous devez faire avec ce que vous avez. Cela veut dire bien sûr que, pour que l'administration systèmes soit redondante, une solution – et c'est parfois la seule – consiste à répartir la charge : d'autres membres du projet prennent chacun une ou deux compétences jusqu'à ce qu'il y ait redondance.

Il n'y a guère de différence entre le côté développement et le côté créatif d'un projet ; si la moitié de votre programme est écrite en C++, l'autre moitié en Python, et qu'un seul développeur sait programmer en Python, son départ du projet provoquera de gros problèmes à court terme et pourrait aussi causer de sérieux problèmes à plus long terme. Encourager les développeurs à se diversifier et à se familiariser avec d'autres langages, paradigmes, bibliothèques, etc. entraîne que chacun de vos développeurs gagne en valeur ; cela ne devrait pas choquer : l'acquisition de nouvelles compétences est le résultat d'un apprentissage qui se poursuit tout au long de la vie, et un personnel mieux formé a aussi plus de valeur. (Cela rend aussi le curriculum vitae de chacun plus attractif, ce qui devrait être une bonne motivation.)

La plupart des développeurs *open source* que je connais considèrent comme un défi et un plaisir de s'aventurer sur de nouveaux territoires : c'est justement ce genre d'état d'esprit qui les a menés à développer de l'*open source* au départ. De même, les administrateurs de systèmes *open source* sont une denrée rare, et ne peuvent se permettre de s'enliser dans une routine. De nouvelles technologies intéressantes pour les administrateurs systèmes apparaissent constamment et il existe souvent de nouvelles façons d'utiliser des technologies actuelles ou anciennes afin de renforcer l'infrastructure ou d'améliorer leur efficacité.

John n'était pas un bon candidat parce qu'il apportait peu de valeur ajoutée ; et il apportait peu de valeur car il n'était jamais allé au-delà des limites du rôle qui lui était attribué. Les administrateurs systèmes *open source* qui tombent dans ce piège ne nuisent pas seulement au projet dans lequel ils sont impliqués sur le moment, ils réduisent leur valeur pour d'autres projets utilisant des technologies d'infrastructure différentes et qui auraient vraiment besoin d'un coup de main ; cela diminue les capacités globales de la communauté *open source*. Pour un administrateur de logiciel libre efficace, il n'existe pas de zone de confort.

9. Des sauvegardes pour votre santé mentale

Austin Appel

*Austin Appel, alias « le cramé », est un professionnel de la sécurité informatique qui passe son temps à fracturer (de façon dûment autorisée, évidemment) des systèmes qu'on croyait sécurisés. On le croise souvent en train d'enseigner l'art de crocheter des serrures durant des conférences de sécurité et de hacking. Dans le monde de l'*open source*, il fait une foule de choses pour le projet Rockbox et a œuvré bénévolement pour le projet* One Laptop Per Child[1] *(un ordinateur portable pour chaque enfant).*

Les sauvegardes, c'est bien. Les sauvegardes, c'est super. Un administrateur compétent fait toujours des sauvegardes régulières. On apprend ça dans n'importe quel manuel traitant de l'administration des serveurs. Le problème, c'est qu'on n'utilise les sauvegardes qu'en cas d'absolue nécessité. Lorsque quelque chose de grave arrive au serveur ou à ses données et qu'on est forcé de se replier sur autre chose, les sauvegardes viendront à point nommé dans une urgence désespérée. Cependant, cela ne devrait jamais arriver, n'est-ce pas ? À tout autre moment, à quoi servent vos sauvegardes, pour vous ou pour votre environnement serveur ?

1 https://fr.wikipedia.org/wiki/One_Laptop_per_Child.

Avant d’aller plus loin, il est important de noter que ce conseil vaut pour les administrateurs serveurs des plus petits projets *open source* – la majorité silencieuse. Si vous maintenez des services qui risquent d’engendrer une grande frustration et peut-être même de causer du tort s’ils tombent et sont indisponibles, vous ne devriez pas prendre ceci pour argent comptant.

Pour le reste d’entre nous qui travaillons sur d’innombrables petits projets avec des ressources limitées, nous n’avons que rarement deux serveurs séparés pour la production et les tests. En vérité, avec tous les services qu’un projet *open source* doit maintenir (système de gestion de version, services web, listes de diffusion, forums, fermes de compilation, bases de données, traqueurs de bogues ou de fonctionnalités, etc.), des environnements de test séparés sont souvent de l’ordre du rêve. Malheureusement, l’approche courante de l’administration systèmes est d’avancer avec précaution et de ne mettre les systèmes à jour que lorsque c’est absolument nécessaire, afin d’éviter tout problème de dépendance, de code cassé, ou n’importe laquelle des millions de choses qui pourraient mal tourner. Si cela vous rend nerveux, ce n’est pas parce que vous manquez éventuellement d’expérience et il est important de savoir que vous n’êtes pas seul dans ce cas. Que nous l’admettions ou non, beaucoup d’entre nous ont été (et sont probablement encore) dans cette situation. C’est triste mais le fait est que cette inaction – découlant de la peur de détruire un système fonctionnel – conduit fréquemment à faire tourner des services qui ont souvent plusieurs versions de retard, au risque d’héberger de nombreuses failles de sécurité potentiellement sérieuses. Pourtant, je vous rassure, ce n’est pas la seule manière de jouer le jeu.

Les gens ont tendance à jouer un jeu différent selon qu’ils ont une infinité de vies ou qu’ils doivent tout recommencer au début dès qu’une seule erreur a été commise. Pourquoi devrait-il en être autrement dans l’administration systèmes ? Aborder le concept de sauvegardes avec un état d’esprit offensif peut changer complète-

ment votre conception de l'administration systèmes. Au lieu de vivre dans la peur d'une commande de `dist-upgrade` complète (ou de son équivalent pour yum, pacman, etc.), celui qui est armé de sauvegardes est libre de mettre à jour les paquets d'un serveur, en toute tranquillité parce qu'il sait qu'il pourra revenir en arrière sur ces modifications si les choses devaient tourner au vinaigre. La clé pour surmonter ses craintes réside tout entière dans l'état d'esprit. Il n'y a aucune raison d'avoir peur dès lors qu'au moment de sauter le pas, vous avez sous la main un filet de sécurité : vos données sauvegardées. Après tout, l'administration systèmes est une expérience d'apprentissage permanente.

Bien sûr, si vous ne validez pas vos sauvegardes, vous reposer dessus devient un jeu très dangereux. Heureusement, les administrateurs systèmes expérimentés savent que le commandement : « Garde des sauvegardes à jour » est toujours suivi par : « Valide tes sauvegardes ». À nouveau, c'est un des mantras que les gens aiment à réciter. Ce que ne dit pas le mantra, dans sa forme élégante et accrocheuse, c'est la rapidité et la facilité avec laquelle il est possible de valider ses sauvegardes. La meilleure manière de savoir si une sauvegarde est fonctionnelle est, bien sûr, de la restaurer (de préférence sur un système identique qui n'est pas en cours d'utilisation). Mais en l'absence d'un tel luxe, on doit à nouveau faire preuve d'un peu plus de créativité. Au moins pour les fichiers, c'est là que les sommes de contrôle peuvent vous aider à vérifier l'intégrité de vos fichiers sauvegardés. Avec `rsync`, par exemple, la méthode utilisée par défaut pour déterminer quels fichiers ont été modifiés consiste à regarder la date et l'heure de la dernière modification, ainsi que la taille du fichier. Cependant, en utilisant l'option `-c`, `rsync` utilisera une somme de contrôle MD4 de 128 bits pour déterminer si les fichiers ont changé ou non. Bien que ce ne soit pas toujours la meilleure idée à mettre en œuvre à chaque fois et en toute occasion – à cause d'un temps d'exécution beaucoup plus long qu'un `rsync` normal et d'une utilisation accrue des accès disques –, cette méthode permet de s'assurer que les fichiers sont intègres.

Le rôle d'un administrateur systèmes peut être éprouvant par moments. Il n'est cependant pas nécessaire de le rendre plus stressant qu'il n'est déjà. Avec le bon état d'esprit, certaines commandes de précaution qui semblent n'avoir qu'un usage unique et limité peuvent être utilisées comme des outils précieux et vous permettre de progresser avec agilité, tout en préservant votre santé mentale et la rapidité que l'on apprécie tant dans les projets *open source*.

Le code

10. L'art de résoudre les problèmes

Thiago Madeira

Thiago Macieira est doublement diplômé. Il a une maîtrise en administration des affaires (MBA) et un diplôme d'ingénieur, mais son implication dans le mouvement open source*, depuis près de 15 ans maintenant, est antérieur à ses diplômes. Participant actif des communautés KDE, Qt et MeeGo, il a été ingénieur logiciel et responsable produit pour Qt, il a fait des conférences et il a écouté les gens. À présent, Thiago vit à Oslo en Norvège et quand il ne travaille pas sur Qt, il essaie, sans grand succès, d'améliorer son skill à StarCraft 2.*

Les problèmes forment une routine à laquelle nous sommes confrontés presque tous les jours ; nous les résolvons et c'est tellement habituel que bien souvent nous n'en avons même pas conscience. Il peut s'agir de situations aussi simples que chercher le meilleur chemin pour arriver à destination ou trouver la meilleure façon de tout faire tenir dans le réfrigérateur. Ce n'est que lorsque nous ne parvenons pas à les résoudre immédiatement que nous remarquons les problèmes car nous devons alors nous arrêter et y réfléchir. Notre vie professionnelle n'échappe pas à cette règle et la résolution de problèmes commence à faire partie de la description du poste à pourvoir.

La résolution de problèmes était le sujet de mon premier cours quand j'ai commencé ma formation d'ingénieur. Dans cet amphithéâtre bondé du siècle dernier, notre professeur expliquait à environ 700 étudiants de première année en quoi les ingénieurs étaient des *solutionneurs* de problèmes et comment nos vies professionnelles consisteraient à enchaîner les problèmes à résoudre. Certains seraient des problèmes faciles résolus en deux temps trois mouvements ; d'autres seraient tellement difficiles que nous aurions besoin d'une structure de projet et d'une équipe pour les résoudre, mais la plupart se situeraient entre ces deux extrêmes. Puis il commença à donner des exemples de la façon dont sa propre mentalité de « solutionneur de problèmes » l'avait aidé dans sa vie professionnelle et personnelle, et nous en offrit même un exemple en direct quand tout à coup le projecteur nous tomba dessus.

La faculté de résoudre des problèmes est un talent que nous pouvons affiner par la pratique et un travail de fond. La pratique est quelque chose que l'on ne peut acquérir que par l'expérience, par succession d'essais et d'erreurs[1] ; ce n'est donc pas quelque chose qu'on peut apprendre dans un livre. Se mettre en situation de résoudre des problèmes, en revanche, est quelque chose que l'on peut apprendre. Face à des problèmes nouveaux, l'expérience est comme notre boîte à outils, et les techniques de résolution sont le mode d'emploi de ces outils.

Formuler correctement la question

La question à laquelle nous essayons de répondre indique la direction que nous allons prendre en essayant de résoudre le problème. Posez la mauvaise question et les réponses seront peu pertinentes, invalides ou simplement complètement fausses. Par conséquent, poser la bonne question est essentiel. De plus, poser correctement la bonne question est important, car cela apporte des indices quant à ce que nous recherchons.

1 http://fr.wikipedia.org/wiki/Apprentissage#Apprentissage_par_essais_et_erreurs.

La manière la plus inutile d'énoncer un problème qu'on puisse rencontrer est : « Ça marche pas », c'est pourtant un grand classique. Certes, l'énoncé est juste puisque manifestement quelque chose a planté. Néanmoins, cette façon de présenter le problème n'apporte aucun indice sur le point de départ pour rechercher des solutions.

Les systèmes de gestion de bogues imposent souvent au rapporteur du bogue de préciser les actions effectuées qui ont conduit à ce problème, la description de ce qui s'est passé — c'est-à-dire le symptôme — et une description du comportement attendu. La comparaison entre le symptôme et le comportement attendu est un bon point de départ pour poser la question fondamentale : « Pourquoi cela s'est-il produit, pourquoi cet autre processus ne s'est-il pas déroulé ? » Même si ce n'est pas la seule manière d'y arriver, appliquer cette technique à des problèmes peut certainement aider à formuler la question.

Formuler correctement le problème et la question, dans ses moindres détails, est aussi une manière de décrire davantage le problème tel qu'il s'est manifesté. En premier lieu, nous devons avoir conscience que le problème ne se trouve probablement pas là où nous nous attendons à le trouver — si c'était le cas, nous l'aurions probablement déjà résolu. Présenter tous les détails du problème permet à l'assistance technique d'avoir plus d'informations pour travailler. De plus, même si c'est contre-intuitif, le fait de décrire le problème dans sa totalité conduit souvent à trouver la solution, si bien que de nombreux groupes de développement ont besoin que des développeurs se concentrent sur cette tâche, soit en discutant avec un collègue, soit en s'adressant à un être innocent, tel qu'un canard en caoutchouc ou M. Patate.

De plus, il faut revenir régulièrement à la question afin de garder l'objectif dans le viseur. Lors de la résolution du problème, il convient de faire attention à ne pas se concentrer exclusivement sur l'une de ses parties au risque de perdre de vue l'objectif global. Pour la même raison, il est nécessaire de reprendre la

question de départ lorsqu'on a trouvé une solution éventuelle et s'assurer ainsi qu'elle couvre bien l'intégralité du problème. Là encore, cela prouve bien la nécessité de poser la bonne question, qui décrira le problème dans son intégralité : si la question est incomplète, la solution pourrait bien l'être également.

Diviser pour mieux régner

L'expérience que j'ai acquise en aidant des utilisateurs en ligne à résoudre leurs problèmes m'a appris que la plupart des personnes considèrent leurs difficultés comme des pierres d'achoppement, monolithiques et indivisibles, qu'il faut traiter comme un tout. Vu sous cet angle, un vaste problème pose une question à laquelle il est très difficile de répondre entièrement.

À vrai dire, la grande majorité de ces problèmes peut se décomposer en plusieurs petits problèmes qu'il est donc plus facile de traiter séparément afin de déterminer s'ils sont la cause originelle du problème, sans parler de la possibilité qu'il y ait plusieurs origines au symptôme rapporté. Répéter cette opération, ne serait-ce qu'un petit nombre de fois, revient à s'attaquer à des problèmes mieux circonscrits, et amène par conséquent à des solutions plus rapides.

Cependant, plus nous sommes obligés de décomposer, mieux nous devons connaître le fonctionnement interne du système que nous avons sous la main. De fait, celui qui doit résoudre un problème ne pourra le décomposer qu'aussi loin que sa connaissance du sujet le lui permettra, et c'est depuis ce point qu'il pourra ensuite traiter la question.

Pour ce qui concerne le développement logiciel, les sous-systèmes utilisés sont souvent de bons indices pour trouver comment décomposer le problème. Par exemple, si le problème implique une transmission de données par TCP/IP, deux subdivisions possibles sont l'expéditeur et le destinataire : il ne sert à rien de chercher le problème du côté du destinataire si l'expéditeur ne

transmet pas les données correctement. De même, une application graphique qui n'affiche pas les données appelées dans une base de données a une division claire : ce serait une bonne idée de vérifier d'abord que l'accès à la base de données fonctionne avant d'enquêter sur la cause du mauvais affichage. Par ailleurs, on pourrait également envoyer des informations quelconques aux fonctions d'affichage et vérifier que ces données s'affichent correctement.

Même quand les regroupements ne sont pas faciles à faire, diviser le problème peut tout de même contribuer à éclairer la question. En fait, les divisions sont presque toujours utiles, car elles réduisent la quantité de code à inspecter et le niveau de complexité à gérer. Au pire, le simple fait de diviser le code en deux parties et de chercher le problème dans l'une des deux peut être utile. Cette technique, nommée bissection, est recommandée si les divisions créées à partir des sous-systèmes et des interfaces n'ont pas encore apporté de solution.

Une succession de divisions appropriées aura pour résultat final un petit exemple autonome qui expose le problème. À ce stade, l'une des trois options suivantes est habituellement la bonne : le problème peut être identifié et localisé ; le code est en fait correct et c'était ce que l'on en attendait qui était faux ; ou bien un bogue a été trouvé dans la couche de code de plus bas niveau. Un des avantages de ce procédé, c'est qu'il génère un scénario de test à joindre à un rapport de bogue, pour peu qu'un bogue soit en cause.

Conditions aux limites

Une question similaire à la division du problème est celle des conditions aux limites. En mathématiques et en physique, les conditions aux limites sont l'ensemble des valeurs qui déterminent la région de validité des équations résolues. Pour le logiciel, les conditions aux limites sont l'ensemble des conditions qui doivent être satisfaites pour que le code s'exécute correctement.

Habituellement, les conditions aux limites sont loin d'être simples : à la différence des mathématiques ou de la physique, les variables des systèmes logiciels sont beaucoup trop nombreuses, ce qui signifie que leurs conditions aux limites sont également légion.

Dans les systèmes logiciels, les conditions aux limites sont souvent nommées « conditions préalables », c'est-à-dire des conditions qui doivent être satisfaites avant qu'une certaine action ne soit autorisée. Quand on est à la recherche d'une réponse, un bon exercice consiste à vérifier que ces conditions préalables ont été satisfaites, car leur violation est clairement un problème qui doit être résolu, même si ce n'est pas la cause première du problème initial. Des conditions préalables peuvent tout simplement se présenter sous la forme d'un pointeur qui doit être valide avant qu'on puisse le déréférencer, ou d'un objet qui ne doit pas être éliminé avant de pouvoir être utilisé. Les conditions préalables complexes seront très probablement documentées en vue de l'utilisation du logiciel.

Un autre groupe intéressant de conditions aux limites se caractérise, curieusement, par ce qui n'est pas autorisé : le comportement indéfini. Ce type de conditions aux limites est très commun lorsque l'on traite des spécifications qui essaient d'être très explicites sur la manière dont le logiciel est censé se comporter. Les compilateurs et les définitions de langage en sont un bon exemple. À strictement parler, déréférencer un pointeur *null* est un comportement indéfini : la conséquence la plus commune en est l'enregistrement d'une exception du processeur et l'arrêt du programme, mais d'autres comportements sont aussi autorisés, y compris le fonctionnement sans faille.

Le bon outil pour le bon usage

Si les ingénieurs sont des *solutionneurs* de problèmes, la devise de l'ingénieur est : « Utilise le bon outil pour le bon usage ». Cela peut sembler évident, étant donné qu'on ne s'attend pas à ce que

quelqu'un utilise un marteau pour résoudre un problème électronique. Cependant, les cas d'utilisation du mauvais outil sont plutôt fréquents, souvent parce qu'on ignore qu'il existe un meilleur outil.

Certains de ces outils sont la base du développement logiciel, comme le compilateur et le débogueur. L'incapacité à se servir de ces outils est impardonnable : le professionnel qui se retrouve dans un environnement d'outils nouveaux ou inconnus — s'il change de poste ou d'emploi, par exemple — doit consacrer du temps à apprendre à les utiliser, à se familiariser avec leurs fonctionnalités et limitations. Par exemple, si un programme plante, être capable de déterminer l'endroit du plantage ainsi que les variables appelées dans cette portion du code peut aider à trouver la cause du problème et donc la solution.

D'autres outils peu connus sont plus évolués, prévus pour des emplois spécifiques, ou encore ne sont disponibles qu'à un prix ou sous des conditions que l'ingénieur ne peut réunir. Ils peuvent toutefois être incroyablement utiles pour contribuer à la résolution de problèmes. Il peut s'agir de vérificateurs de code statique ou de processus, de débogueurs de mémoire, d'enregistreurs d'évènements matériels, etc. Par exemple, le matériel de développement inclut souvent un système qui permet de le contrôler à l'aide d'une interface dédiée comme JTAG ou de lister toutes les instructions exécutées et l'état des processeurs. Mais cela nécessite d'avoir du matériel et des outils spécifiques, qui ne sont pas facilement accessibles et coûtent plus cher que les machines et périphériques grand public. Un autre exemple est la suite d'outils Valgrind[1], qui comprend un vérificateur de processus et des débogueurs de mémoire. L'ensemble est gratuit, facilement disponible, mais fait partie de ces outils spécifiques de haut niveau dont l'usage n'est pas enseigné à l'école.

Connaître le contenu de sa boîte à outils est un savoir précieux. L'utilisation d'un outil spécialisé pour chercher un problème va

1 http://fr.wikipedia.org/wiki/Valgrind.

probablement donner un résultat plus rapide, qu'il soit positif et confirme le problème, ou négatif, et oriente la recherche dans une autre direction. Par ailleurs, il est important de savoir comment utiliser ces outils, ce qui justifie le temps passé à lire la documentation, à s'entraîner ou simplement à expérimenter ces outils avec des problèmes connus pour comprendre comment ils fonctionnent.

Conclusion

La résolution de problèmes est un art accessible à tous. Comme dans tous les arts, certains semblent avoir une telle facilité qu'ils semblent être nés avec cette compétence. Mais en réalité, avec assez d'expérience et de pratique, résoudre des problèmes devient une activité inconsciente.

Quand on est confronté à un problème qui n'est pas évident à résoudre, il faut s'asseoir et le considérer dans son intégralité. Quel problème avons-nous ? Pouvons-nous formuler la question à laquelle nous devons répondre ? Une fois que nous savons ce que nous cherchons, nous pouvons commencer à examiner où peut être situé le problème. Peut-on le décomposer en parties plus petites et plus maniables ? Quels sont les meilleurs outils à utiliser pour chaque partie ? Avons-nous vérifié que nous utilisions correctement les fonctionnalités et services disponibles ?

Après avoir résolu un certain nombre de problèmes, on commence à repérer des schémas. Il devient plus facile de détecter des indices subtils à partir des symptômes et de diriger les recherches vers le problème réel. Un solutionneur de problèmes expérimenté peut ne pas même se rendre compte que cette action a lieu. C'est un signe que l'expérience et les automatismes se sont si bien mis en place qu'il n'y a plus besoin d'effort conscient pour accéder à ces compétences.

Bien sûr il, y aura toujours des problèmes qui seront difficiles à résoudre dans la vie : problèmes professionnels, existentiels,

philosophiques ou même ceux qui sont causés par la pure curiosité. Là encore, c'est le défi qui nous stimule, le besoin de comprendre toujours plus et mieux. Sans cela, la vie serait trop triste.

11. La collaboration entre projets

Henri Bergius

Henri Bergius est le fondateur de Midgard[1]*, un dépôt de contenu pour les logiciels libres. Il s'est aussi longtemps investi dans la géolocalisation d'ordinateurs de bureau sous Linux ainsi que dans les communautés Maemo et MeeGo. Il dirige un petit cabinet de conseil nommé Nemein, bidouille en CoffeeScript et PHP et passe une grande partie de son temps libre à faire de la moto dans des régions reculées du continent Eurasien. Il vit dans la froide ville nordique d'Helsinki, en Finlande.*

> « Il peut y avoir un système complètement nouveau dans lequel vous êtes défini non par ce que vous possédez mais davantage par qui vous êtes, par ce que vous avez créé et partagé, par ce que d'autres personnes auront ensuite construit sur cette base. »
>
> — John Seely Brown, ancien directeur de Xerox PARC dans *An Optimist's Tour of the Future* (Mark Stevenson, 2010)

Des projets et des communautés

Le monde du logiciel libre est pour l'essentiel divisé en tribus rassemblées autour de choses qu'on appelle des *projets*. Il existe des projets majeurs tels que GNOME, KDE ou Drupal et il existe

1 http://midgard-project.org.

bien d'autres projets plus modestes qui tournent autour d'une seule application ou bibliothèque logicielle.

En fait, les qualifier de « projets » est un peu ridicule.

Selon moi, un projet est un effort organisé et tendu vers un but réalisable avec un calendrier qui inclut des dates de début et de fin. Ainsi, GNOME 3.1 serait par exemple un projet tandis que GNOME, pris dans son ensemble, n'en est pas un. Il s'agit d'une communauté d'individus qui entretiennent et créent le corps d'un logiciel par de multiples petits efforts, ou projets.

Trève de pédanterie. Le souci avec le concept de projets, c'est qu'il finit par tenir des personnes à l'écart. Cela crée des communautés isolées qui sont souvent réticentes à collaborer avec « la concurrence », ou qui en sont même incapables. Pourtant en fin de compte, toutes ces communautés sont composées de personnes qui écrivent des logiciels libres, et ce sont elles qui décident s'il est possible ou non d'utiliser un logiciel dans différents environnements.

Finalement, nous avons tous le désir de créer un logiciel qui sera utilisé par d'autres. Mieux encore : nous avons envie que les autres joignent leurs efforts aux nôtres et créent des trucs super à partir de ce que nous avons créé. Après tout, c'est ce qui forme le cœur même des logiciels libres.

Alors pourquoi érigeons-nous ces murs autour de nous ? Garder une communauté isolée ne fait que favoriser une mentalité de type « nous contre eux ». Les incompatibilités entre les différents langages de programmation contribuent déjà fortement à notre division. Pourquoi en rajouter ?

La leçon de Midgard

Il y a une chose que j'aurais aimé savoir quand j'ai démarré, dans cette période optimiste des « .com » de la fin des années 90, c'est qu'en réalité le développement de logiciels n'a pas besoin de

s’isoler. Avec quelques précautions, nous pouvons partager nos logiciels et nos idées avec des communautés, et cela renforce et améliore à la fois les logiciels et les communautés.

Quand j’ai démarré ma carrière dans le logiciel libre, c’était l’époque des grands projets. Netscape ouvrait son code source, la fondation Apache prenait forme et des fonds de capital-risque se levaient un peu partout. Tenter de bâtir sa communauté devenait la norme. C’était le chemin assuré vers la gloire, la fortune et la réalisation de choses extraordinaires.

Alors nous avons construit notre propre cadre d’applications web. À ce moment-là, il n’y en avait pas tant que cela, en particulier pour le tout jeune langage qu’était le PHP. Le PHP n’était même pas notre premier choix. Nous l’avions seulement choisi au terme d’un long débat sur le fait d’utiliser Scheme[1] que notre développeur principal préférait. Mais le PHP gagnait alors en popularité et devenait le langage de programmation du Web. Et c’était bien le Web que nous voulions construire.

Au début, les choses semblaient prometteuses. Beaucoup de développeurs rejoignaient notre communauté et commençaient à y contribuer. Il y a même eu des entreprises fondées autour de Midgard. Notre cadre d’applications gagnait en fonctionnalités et devenait plus étroitement associée à Midgard.

Avec le recul, c’est là que nous avons fait une erreur. Nous avions positionné Midgard pour être distinct du PHP lui-même, comme quelque chose que vous installeriez séparément et utiliseriez comme base pour construire des sites entiers. Il fallait suivre notre voie. C’était ça ou rien.

Avec Midgard, vous deviez utiliser notre interface de dépôt de contenus pour tout, ainsi que notre modèle de permissions et de gestion des utilisateurs. Vous deviez utiliser notre système de gabarits et stocker une grande partie de votre code dans le dépôt au lieu d’utiliser un système de fichiers.

1 http://fr.wikipedia.org/wiki/Scheme.

Ceci ne passait évidemment pas très bien auprès de l'ensemble de la communauté PHP. Nos idées leur semblaient étranges, et Midgard, à ce moment-là, était même distribué en tant que gigantesque correctif du code base puisqu'on ne pouvait pas charger de modules avec PHP3.

Les années ont passé et la popularité de PHP a connu des hauts et des bas. Pendant ce temps, la communauté Midgard est restée relativement constante : un petit groupe soudé faisant des progrès sur le long terme mais séparé du monde plus large de PHP.

Nous nous sommes toujours demandé pourquoi il était si difficile d'interagir avec le monde PHP. Même des communautés qui faisaient des choses complètement différentes, comme l'environnement de bureau GNOME, semblaient plus faciles à approcher. Ce n'est que récemment, après des années d'isolement, que nous avons pris conscience du problème. En résumé : les cadres d'applications nous séparent alors que les bibliothèques nous permettent de partager notre code et nos expériences.

À propos des bibliothèques et des infrastructures

En définitive, les logiciels ont pour objectif l'automatisation, la construction d'outils que les autres peuvent utiliser pour résoudre des problèmes ou se connecter entre eux. Avec les logiciels, ces outils comportent plusieurs couches. Il existe des services de bas niveau comme les systèmes d'exploitation, puis les bibliothèques, les infrastructures, les boîtes à outils et enfin les applications elles-mêmes.

Les applications sont toujours écrites pour des usages spécifiques, donc entre elles il existe peu de possibilités de partage de code.

Les possibilités les plus séduisantes se situent au niveau des bibliothèques et infrastructures. Une infrastructure, si elle est suffisamment générique, peut généralement être utilisée pour construire différentes sortes de logiciels. Une bibliothèque, quant

à elle, peut être utilisée pour apporter un élément particulier de logique ou de connectivité là où le besoin s'en fait sentir. De mon point de vue, c'est dans cette couche que le plus gros de la programmation devrait être fait, avec des applications spécifiques qui ne sont que des connexions entre diverses bibliothèques au sein d'une infrastructure qui s'occupe alors de faire tourner l'application en question.

Qu'est-ce qu'une bibliothèque et qu'est-ce qu'une infrastructure ? Les gens utilisent souvent ces termes indifféremment bien qu'il existe une règle grossière qui permet de les différencier : une bibliothèque est une ressource à laquelle votre code fait appel, alors qu'une infrastructure est une ressource qui fait appel à votre code.

Si vous voulez que votre code soit utilisé et amélioré, le meilleur moyen est de maximiser le nombre de ses utilisateurs et contributeurs potentiels. Avec le logiciel libre, cela fonctionne en s'assurant que votre code peut être adapté à de multiples situations et environnements.

En définitive, ce que vous voulez c'est développer une bibliothèque. Les bibliothèques, c'est cool.

Comment faire en sorte que la collaboration fonctionne

Le plus difficile est de franchir la barrière du « eux-contre-nous ». Les développeurs de l'autre communauté sont des bidouilleurs concevant du logiciel libre, tout comme vous. Il suffit donc de franchir le pas et de commencer à leur parler.

Une fois le débat engagé, voici quelques points que j'ai trouvés importants quant à l'application effective des idées communes ou des bibliothèques au-delà des frontières du projet :

– Utilisez des licences permissives et essayez d'éviter les cessions de droits d'auteurs et autres exigences que les utilisateurs potentiels trouveraient onéreuses. Hébergez le

projet en terrain neutre. Pour les projets web, Apache est un assez bon havre. Pour les projets bureautiques, Freedesktop est probablement le meilleur choix. Utilisez des technologies qui n'imposent pas trop de contraintes. Les bibliothèques doivent être de bas niveau, ou fournir des API (interfaces de programmation) D-Bus utilisables avec n'importe quel système ;

– Évitez les dépendances spécifiques à une infrastructure. KDE a, par exemple, trouvé GeoClue difficile à adopter parce qu'il utilise des paramètres spécifiques à l'interface GNOME. Rencontrez les autres. Si vous venez du projet GNOME, allez à l'aKademy et donnez-y une conférence. Si vous êtes développeur KDE, allez parler au GUADEC. Après avoir partagé une bière ou deux, la collaboration par IRC vient beaucoup plus naturellement ;
– Enfin, vous devez accepter que votre implémentation ne soit pas utilisée par tout le monde. Mais si, au moins, d'autres mettent en œuvre les mêmes idées, alors une collaboration reste possible.

Bonne chance pour abattre les frontières du projet ! Dans la plupart des cas, cela fonctionne si vos idées sont bonnes et présentées avec un esprit ouvert. Mais même si vous ne trouvez pas de terrain d'entente, tant que votre application résout le problème que vous vous posiez de façon satisfaisante, vous n'aurez pas œuvré en vain. Après tout, ce qui compte c'est de proposer des logiciels et d'offrir la meilleure expérience utilisateur possible.

12. L’écriture de correctifs

Kai Blin

Kai Blin est un bio-informaticien[1] *qui mène des recherches sur les antibiotiques dans le cadre de ses activités quotidiennes, tant sur ordinateur qu’au labo. Il est très heureux de pouvoir diffuser le logiciel développé dans le cadre de ses activités professionnelles sous licence* open source*. Vivant dans la charmante ville de Tübingen, dans le sud de l’Allemagne, Kai passe une partie de ses soirées sur son ordinateur à programmer pour le projet Samba. Il consacre la majeure partie de son temps libre au théâtre, où il participe aussi bien à la performance scénique qu’à la construction d’accessoires ou à la régie technique dans les coulisses.*

La première interaction concrète que vous pouvez avoir avec un projet *open source* consiste souvent à écrire des correctifs. C’est la première impression que vous donnez aux développeurs présents. Proposer de « bons » premiers correctifs, ou tout du moins jugés comme tels par le projet auquel vous contribuez, vous rendra la vie plus facile. Les règles précises d’écriture du correctif, la façon de le soumettre au projet et tous les autres détails nécessaires vont sans doute varier selon les divers projets auxquels vous voulez contribuer. Mais j’ai trouvé quelques règles générales que l’on retrouve presque à chaque fois. Et c’est ce dont traite cet article.

1 http://fr.wikipedia.org/wiki/Bio-informatique.

Comment tout foirer

Le fil rouge de ce livre est « ce que j'aurais aimé savoir avant de commencer », aussi permettez-moi de commencer par l'histoire de mes premiers correctifs. J'ai été impliqué sérieusement dans l'écriture de code pour la première fois pendant le Google Summer of Code[1] (GSoC) de 2005. Le projet Wine[2] avait accepté que j'implémente un chiffrement NTLM basé sur des outils connexes à Samba. Wine est un projet à *committer* unique, ce qui signifie que seul le développeur principal, Alexandre Julliard, possède les autorisations de commit sur le dépôt principal. En 2005, Wine utilisait encore CVS comme système de gestion de versions. Quand le projet a démarré et que j'ai reçu le courriel me disant que j'étais accepté, j'ai contacté mon mentor sur IRC et me suis mis au travail.

J'alignais joyeusement les lignes de code et bientôt j'ai pu implémenter les premières fonctionnalités. J'ai produit un correctif et l'ai soumis à mon mentor pour qu'il fasse une relecture. Au temps du CVS, il fallait renseigner toutes les options de diff[3] manuellement, mais je m'étais particulièrement documenté sur la partie cvs et j'ai fait `diff -N -u > ntlm.patch`. De cette façon, j'avais le fichier que je pouvais envoyer à mon mentor. En fait, voilà une chose que j'ai effectuée correctement. Et c'est la première chose que vous devriez prendre en compte quand vous préparez un correctif. Le résultat classique de la commande diff est sans doute plus facile à lire pour un ordinateur, mais je n'ai jamais rencontré un être humain qui préfère le résultat classique au résultat d'un diff unifié. Grâce à l'option -u , diff utilise les notations +++ et `---`.

Par exemple, le diff qui suit est le résultat de la réécriture de « Hello, World! » en Python, version suédoise.

1 https://en.wikipedia.org/wiki/Google_Summer_of_Code.
2 http://www.winehq.org.
3 Un *diff* (abréviation de différence) est un fichier qui affiche le résultat d'une comparaison entre deux éléments (en général, des lignes de code). Pour en savoir plus http://fr.wikipedia.org/wiki/Diff.

```
diff–git a/hello.py b/hello.py index
59dbef8..6334aa2 100644
--- a/hello.py
+++ b/hello.py @@ -1,4 +1,4 @@
#!/usr/bin/env python
# vim: set fileencoding=utf-8
-print "Hello, world!"
+print "Halla, varlden!"
```

La ligne qui commence par « - » est la ligne supprimée, celle qui commence par « + » est celle qui va être ajoutée. Les autres lignes permettent à l'outil de correction de faire son travail.

J'ai envoyé le diff unifié que je venais de créer à mon mentor qui m'en a fait une *review*[1] en me signalant beaucoup d'éléments à modifier. J'ai effectué les corrections et lui ai renvoyé un nouveau *diff* peu de temps après. Le cycle d'analyse du code a continué pendant toute la durée du GSoC et mon correctif ne cessait de grossir. Quand la date de livraison est arrivée, je me suis retrouvé avec un correctif monstrueux dans lequel étaient inclus tous mes changements. Naturellement, j'ai eu beaucoup de mal à obtenir une review de mon correctif, sans parler de le faire accepter. Pour finir, Alexandre refusa de regarder plus avant le correctif tant que je ne l'aurais pas scindé. Les règles en usage chez Wine exigent que les correctifs soient de petites étapes logiques ajoutant une fonctionnalité. Chaque correctif doit faire quelque chose d'utile et pouvoir être compilé.

De fait, scinder un énorme correctif existant en différentes parties cohérentes individuellement et qui peuvent être compilées représente beaucoup de travail. C'était même d'autant plus de travail que je ne connaissais qu'une seule manière de le faire : écrire un petit correctif, créer le diff, le proposer à la validation, mettre à jour ma contribution locale et écrire alors le correctif suivant. Peu de temps après que j'ai commencé à envoyer mes premiers petits correctifs, Wine est entré dans une phase de gel

1 [NdT] révision minutieuse.

des fonctionnalités d'un mois menant à la version 0.9.0 bêta. Je suis resté sur le correctif suivant pendant un mois avant de pouvoir continuer et j'ai finalement envoyé mon dernier correctif en novembre. Complètement frustré par cette expérience, j'ai décidé que je ne voulais plus jamais avoir affaire à la communauté Wine.

Ma frustration perdura jusqu'à ce que des personnes qui utilisaient réellement mon code commencent à me poser des questions sur celui-ci en février 2006. Mon code était vraiment utile ! Ils voulaient également davantage de fonctionnalités. Quand Google annonça qu'il reconduirait le GSoC en 2006, mes projets pour l'été étaient clairs. Comme Wine avait basculé de diff à git en décembre 2005, je savais que je ne serais pas ennuyé par des gels de fonctionnalités, puisque je pouvais finalement créer tous mes petits correctifs localement. La vie était belle.

C'est seulement quand je suis tombé sur une interface de git (appelée porcelaine dans le langage git) qui émulait le comportement de Quilt[1] que j'ai appris l'existence d'autres outils qui auraient pu me faciliter la vie, même en 2005.

Comment NE PAS tout foirer

Maintenant que je vous ai raconté comment j'ai réussi à me planter avec l'envoi de correctifs, permettez-moi de poursuivre avec quelques conseils pour éviter les pièges.

Règles pour la soumission de correctifs

Mon premier conseil est de lire attentivement toutes les directives de soumission de correctifs que peut avoir le projet auquel vous voulez contribuer. Celles-ci, ainsi que les normes de style de codage, doivent être consultées avant même de commencer à coder.

1 http://en.wikipedia.org/wiki/Quilt_(software).

Des diffs unifiés

Même si ce n'est pas explicitement indiqué dans les directives de soumission des correctifs, vous devez vraiment – oui, vraiment – envoyer le résultat d'un diff unifié. Je n'ai encore rencontré aucun projet qui préfère la sortie non unifiée du diff. Les diffs unifiés rendent la révision du correctif beaucoup plus facile. Ce n'est pas par hasard que les programmes modernes de gestion de versions utilisent automatiquement ce format dans leurs commandes diff par défaut.

Utilisez un contrôle de version distribué

En ce qui concerne la gestion des versions, vous devriez vraiment utiliser un système de gestion de versions distribué (DVCS) pour travailler localement sur le code. Git[1] et Mercurial[2] sont les choix les plus populaires de nos jours, quoique Bazaar[3] puisse aussi très bien faire l'affaire. Même si le projet auquel vous voulez contribuer utilise toujours un système de gestion de version centralisé, être en mesure d'envoyer vos changements de manière itérative est une très bonne chose. Tous les outils de gestion de versions distribués mentionnés ci-dessus devraient être capables d'importer des changements depuis un SVN ou un CVS. Rien ne vous empêche d'apprendre Quilt mais, sérieusement, le futur passe par les systèmes de gestion de versions distribués.

De petits correctifs, pour faire une chose à la fois

Quand je dois examiner des correctifs, les plus ennuyeux à traiter sont ceux qui sont trop gros ou qui tentent de faire de nombreuses choses en même temps. Les correctifs qui ne font qu'une chose à la fois sont plus faciles à traiter. Au final, ils vous faciliteront la vie quand vous devrez déboguer les erreurs qu'auront manquées à la fois l'auteur et le vérificateur.

---

1 http://fr.wikipedia.org/wiki/Git.
2 http://fr.wikipedia.org/wiki/Mercurial.
3 http://fr.wikipedia.org/wiki/Bazaar_(logiciel).

Suivez votre correctif

Après avoir proposé votre correctif, gardez un œil sur les canaux de communication du projet et sur votre correctif. Si vous n'avez eu aucun retour au bout d'une semaine, vous devriez poliment en demander un. En fonction de la façon dont le projet gère les propositions de correctif, celui-ci pourrait être noyé dans le bruit. N'espérez pas voir votre correctif retenu du premier coup. Il faut, en général, quelques essais pour s'habituer au style d'un nouveau projet. En tant que contributeur néophyte, personne ne vous en voudra pour ça, à condition d'avoir presque tout bon. Assurez-vous seulement de corriger ce que les développeurs vous ont indiqué et envoyez une seconde version du correctif.

Conclusion

Écrire de bons correctifs n'est pas difficile. Il y a deux ou trois choses à prendre en considération. Mais après en avoir écrit quelques-uns, vous devriez être au point. Un système moderne de contrôle de version (distribué) et le workflow qui en résulte géreront de fait la plupart des choses que j'ai mentionnées. Si vous n'avez qu'une chose à retenir, c'est ceci :

- utilisez un système de contrôle de version distribué pour gérer vos correctifs ;
- écrivez vos correctifs en petites étapes indépendantes ;
- suivez les normes de codage en vigueur ;
- répondez rapidement aux commentaires sur vos correctifs.

Les quelques lignes directrices ci-dessus devraient vous aider à bien faire la plupart des choses, sinon toutes, quand vous soumettrez vos premiers correctifs. Bon codage.

Assurance qualité

13. Des tests contre les bogues

Même en multipliant les regards, les bogues ne sautent pas aux yeux

Ara Pulido

Ara Pulido est ingénieure d'essais pour Canonical, d'abord comme membre de l'équipe assurance qualité d'Ubuntu, et maintenant dans le cadre de l'équipe de certification du matériel. Même si elle a commencé sa carrière en tant que développeuse, elle a vite découvert que ce qu'elle aimait vraiment, c'était tester les logiciels. Elle est très intéressée par les nouvelles techniques d'analyse et tente d'utiliser son savoir-faire pour améliorer Ubuntu.

Les tests maison ne suffisent pas

Je me suis impliquée dans le logiciel libre dès le début de mes études à l'Université de Grenade. Là-bas, avec des amis, nous avons fondé un groupe local d'utilisateurs de Linux et organisé plusieurs actions pour promouvoir le logiciel libre. Mais, à partir du moment où j'ai quitté l'université, et jusqu'à ce que je commence à travailler chez Canonical, ma carrière professionnelle s'est déroulée dans l'industrie du logiciel propriétaire, d'abord comme développeuse puis comme testeuse.

Lorsque l’on travaille au sein d’un projet de logiciel propriétaire, les ressources pour tester sont très limitées. Une petite équipe reprend le travail initié par les développeurs avec les tests unitaires, utilisant leur expérience pour trouver autant de bogues que possible afin de mettre à disposition de l’utilisateur final un produit aussi abouti que possible. Dans le monde du logiciel libre, en revanche, tout est différent.

Lors de mon embauche chez Canonical, hormis la réalisation de mon rêve d’avoir un travail rémunéré au sein d’un projet de logiciel libre, j’ai été émerveillée par les possibilités des activités de test dans le cadre d’un tel projet. Le développement du produit s’effectue de manière ouverte, et les utilisateurs ont accès au logiciel dès son commencement, ils le testent et font des rapports de bogues dès que c’est nécessaire. C’est un nouveau monde rempli de beaucoup de possibilités pour une personne passionnée par les tests. Je voulais en profiter au maximum.

Comme beaucoup de personnes, je pensais que les tests « maison », c’est-à-dire l’utilisation par soi-même du logiciel que l’on envisage de mettre à disposition, était l’activité de test la plus importante qu’on puisse mener dans le logiciel libre. Mais si, selon la formule de Raymond dans *La cathédrale et le bazar* « avec suffisamment d’observateurs, tous les bogues sautent aux yeux », alors comment se fait-il qu’avec ses millions d’utilisateurs Ubuntu comporte encore des bogues sérieux à chaque nouvelle version ?

La première chose dont je me suis aperçue quand j’ai commencé à travailler chez Canonical c’est que les activités de test organisées étaient rares ou inexistantes. Les seules sessions de test qui étaient d’une certaine façon organisées se présentaient sous la forme de messages électroniques envoyés à une liste de diffusion, manière de battre le rappel pour tester un paquetage dans la version de développement d’Ubuntu. Je ne pense pas que cela puisse être considéré comme une vraie procédure de test, mais simplement comme une autre forme de « test maison ».

Cette sorte de test génère beaucoup de doublons, car un bogue facile à débusquer sera documenté par des centaines de personnes. Malheureusement le bogue potentiellement critique, vraiment difficile à trouver, a de bonnes chances de passer inaperçu en raison du bruit créé par les autres bogues, et ce même si quelqu'un l'a documenté.

En progrès

La situation s'améliore-t-elle ? Sommes-nous devenus plus efficaces pour les tests au sein des projets de développement libre ? Oui, j'en suis convaincue.

Pendant les derniers cycles de développement d'Ubuntu, nous avons commencé bon nombre de sessions de test. La gamme des objectifs pour ces sessions est large, elle comprend des domaines comme de nouvelles fonctionnalités de bureau, des tests de régression, des tests de pilotes X.org ou des tests de matériel d'ordinateur portable. Les résultats sont toujours suivis et ils s'avèrent vraiment utiles pour les développeurs, car ils leur permettent de savoir si les nouveautés fonctionnent correctement, au lieu de supposer qu'elles fonctionnent correctement à cause de l'absence de bogues.

En ce qui concerne les outils d'assistance aux tests, beaucoup d'améliorations ont été apportées :

- Apport[1] a contribué à augmenter le niveau de détail des bogues signalés concernant Ubuntu : les rapports de plantage incluent toutes les informations de débogage et leurs doublons sont débusqués puis marqués comme tels ; les utilisateurs peuvent signaler des bogues sur base de symptômes, etc. ;
- Le Launchpad[2], avec ses connexions en amont, a permis d'avoir une vue complète des bogues, sachant que les

1 http://wiki.ubuntu.com/Apport.
2 http://launchpad.net.

bogues qui se produisent dans Ubuntu se situent généralement dans les projets en amont, et permet aux développeurs de savoir si les bogues sont en cours de résolution ;
- Firefox, grâce à son programme et à son extension Test Pilot[1], mène des tests sans qu'on ait à quitter le navigateur. C'est, à mon sens, une bien meilleure façon de rallier des testeurs qu'une liste de diffusion ou un canal IRC ;
- L'équipe Assurance Qualité d'Ubuntu[2] teste le bureau en mode automatique et rend compte des résultats toutes les semaines, ce qui permet aux développeurs de vérifier très rapidement qu'il n'y a pas eu de régression majeure pendant le développement.

Cependant, malgré l'amélioration des tests dans les projets de logiciel libre il reste encore beaucoup à faire.

Pour aller plus loin

Les tests nécessitent une grande expertise, mais sont encore considérés au sein de la communauté du logiciel libre comme une tâche ne demandant pas beaucoup d'efforts. L'une des raisons pourrait être que la manière dont on les réalise est vraiment dépassée et ne rend pas compte de la complexité croissante du monde du logiciel libre durant la dernière décennie. Comment est-il possible que, malgré la quantité d'innovations amenées par les communautés du logiciel libre, les tests soient encore réalisés comme dans les années 80 ? Il faut nous rendre à l'évidence, les scénarios de tests sont ennuyeux et vite obsolètes. Comment faire grandir une communauté de testeurs supposée trouver des bogues avérés si sa tâche principale est de mettre à jour les scénarios de test ?

---

1 http://testpilot.mozillalabs.com.
2 http://reports.qa.ubuntu.com/reports/desktop-testing/natty.

Mais comment améliorer la procédure de test ? Bien sûr, nous ne pouvons pas nous débarrasser des scénarios de test, mais nous devons changer la façon dont nous les créons et les mettons à jour. Nos testeurs et nos utilisateurs sont intelligents, alors pourquoi créer des scripts pas-à-pas ? Ils pourraient aisément être remplacés par une procédure de test automatique. Définissons plutôt une liste de tâches que l'on réalise avec l'application, et certaines caractéristiques qu'elle devrait posséder. Par exemple, « l'ordre des raccourcis dans le lanceur doit pouvoir être modifié », ou « le démarrage de LibreOffice est rapide ». Les testeurs trouveront un moyen de le faire, et créeront des scénarios de test en parallèle des leurs.

Mais ce n'est pas suffisant, nous avons besoin de meilleurs outils pour aider les testeurs à savoir ce qu'ils testent, où et comment. Pourquoi ne pas avoir des API (interfaces de programmation) qui permettent aux développeurs d'envoyer des messages aux testeurs à propos des nouvelles fonctionnalités ou des mises à jour qui doivent être testées ? Pourquoi pas une application qui nous indique quelle partie du système doit être testée ? En fonction des tests en cours ? Dans le cas d'Ubuntu, nous avons les informations dans le Launchpad (il nous faudrait aussi des données sur les tests, mais au moins nous avons des données sur les bogues). Si je veux démarrer une session de test d'un composant en particulier j'apprécierais vraiment de savoir quelles zones n'ont pas encore été testées ainsi qu'une liste des cinq bogues comptant le plus de doublons pour cette version en particulier afin d'éviter de les documenter une fois de plus. J'aimerais avoir toutes ces informations sans avoir à quitter le bureau que je suis en train de tester. C'est quelque chose que Firefox a initié avec Test Pilot, bien qu'actuellement l'équipe rassemble principalement les données sur l'activité du navigateur.

La communication entre l'amont et l'aval et vice-versa doit aussi être améliorée. Pendant le développement d'une distribution, un bon nombre des versions amont sont également en cours

de développement, et ont déjà une liste des bogues connus. Si je suis un testeur de Firefox sous Ubuntu, j'aimerais avoir une liste des bogues connus aussitôt que le nouveau paquet est poussé dans le dépôt. Cela pourrait se faire à l'aide d'une syntaxe reconnue pour les notes de versions, syntaxe qui pourrait ensuite être facilement analysée. Les rapports de bogue seraient automatiquement remplis et reliés aux bogues amont. Encore une fois, le testeur devrait avoir facilement accès à ces informations, sans quitter son environnement de travail habituel.

Les tests, s'ils étaient réalisés de cette manière, permettraient au testeur de se concentrer sur les choses qui comptent vraiment et font de la procédure de test une activité qualifiée : se concentrer sur les bogues cachés qui n'ont pas encore été découverts, sur les configurations et environnements spéciaux, sur la création de nouvelles manières de casser le logiciel. Et, *in fine*, s'amuser en testant.

Récapitulons

Pour ce que j'en ai vu ces trois dernières années, les tests ont beaucoup progressé au sein d'Ubuntu et des autres projets de logiciels libres dans lesquels je suis plus ou moins impliquée, mais ce n'est pas suffisant. Si nous voulons vraiment améliorer la qualité du logiciel libre, nous devons commencer à investir dans les tests et innover dans la manière de les conduire, de la même façon que nous investissons dans le développement. Nous ne pouvons pas tester le logiciel du XXIe siècle avec les techniques du XXe siècle. Nous devons réagir. Qu'il soit *open source* ne suffit plus à prouver qu'un logiciel libre est de bonne qualité. Le logiciel libre sera bon parce qu'il est *open source* et de la meilleure qualité que nous puissions offrir.

14. Remue-ménage dans le triage

Andre Klapper

Dans la vraie vie, Andre Klapper est maître ès débogage. Pendant sa pause déjeuner ou sa sieste, il travaille à divers trucs sur GNOME (bugsquad, équipe de release, traduction, documentation, etc.), ou Maemo, étudie ou mange de la crème glacée.

Au tout début, je n'avais qu'une seule et unique question : comment imprimer seulement une partie du courriel que j'ai reçu dans *Evolution*, le client de messagerie GNOME ? J'ai donc demandé sur la liste de diffusion.

Ça faisait exactement un an que j'étais passé sous Linux, frustré de ne pouvoir faire fonctionner mon modem après avoir réinstallé un OS propriétaire plutôt populaire à l'époque.

La réponse à ma question fut : « impossible ». Des petits génies auraient parcouru le code, l'auraient compilé, l'auraient bidouillé pour qu'il se comporte comme voulu, puis auraient soumis un correctif joint au rapport de bogue. Bon. Comme vous l'aurez deviné, je n'étais pas un petit génie. Mes talents de programmeur sont plutôt limités, donc sur le moment je suis resté coincé sur une solution de contournement plutôt lourde pour mon impres-

sion. La réponse que j'avais reçue sur la liste de diffusion signalait également que cette fonctionnalité était prévue, et qu'on avait complété pour moi un rapport de bogue – sans préciser où, mais je m'en fichais, j'étais content d'apprendre qu'il était prévu de corriger mon problème prochainement.

Il se peut que je sois resté abonné à la liste de diffusion par simple paresse. Certains mentionnaient le rapporteur de bogues de temps en temps, souvent comme une réponse directe aux demandes de fonctionnalités, alors j'y ai finalement jeté un coup d'œil. Mais les rapporteurs de bogue, en particulier Bugzilla[1], sont d'étranges outils avec beaucoup d'options complexes. Un domaine que vous préférez normalement éviter à moins que vous ne soyez masochiste. Ils contiennent maints tickets décrivant des bogues ou des demandes de fonctionnalités émanant d'utilisateurs et de développeurs. Il semblait également que ces rapports aient été en partie utilisés pour planifier les priorités (appeler cela « gestion de projet » aurait été un euphémisme ; moins d'un quart des problèmes qui étaient planifiés pour être résolus ou implémentés dans une version spécifique étaient réellement corrigés au bout du compte).

Au-delà d'une vision intéressante sur les problèmes du logiciel et sur la popularité de certaines demandes, ce que j'ai découvert, c'est beaucoup de choses incohérentes et pas mal de bruit, comme des doublons ou des rapports de bogues manquant d'éléments pour pouvoir être traités correctement. J'ai eu envie de nettoyer un peu en « triant » les rapports de bogues disponibles. Je ne sais pas bien ce que cela vous dit sur mon état d'esprit – ajouter ici des mots-clés bidon pour une caractérisation aléatoire, comme *organisé, persévérant et intelligent*. C'est assez ironique quand on pense à mon père qui se plaignait toujours du bordel dans ma chambre. Donc à cette époque lointaine de modems commutés, j'avais pour habitude de rassembler mes questions et de les faire remonter sur IRC une fois par jour afin de mitrailler de questions

1 http://www.bugzilla.org.

le responsable des bogues d'Evolution, qui était toujours accueillant, patient et soucieux de partager son expérience. Si jamais à l'époque il y avait un guide de triage qui couvrait les savoirs de base pour la gestion des bogues et qui exposait les bonnes pratiques et les pièges les plus courants, je n'en avais pas entendu parler.

Le nombre de signalements baissa de 20 % en quelques mois, bien que ce ne fût bien évidemment pas grâce à une unique personne faisant le tri des tickets. Il y avait manifestement du travail en attente, tel que diminuer le nombre des tickets attribués aux développeurs pour qu'ils puissent mieux se concentrer, parler avec eux, définir les priorités, et répondre aux commentaires non-traités de certains utilisateurs. L'*open source* accueille toujours bien les contributions une fois que vous avez trouvé votre créneau.

Bien plus tard, j'ai pris conscience qu'il y avait de la documentation à consulter. Luis Villa, qui fut probablement le premier des experts en bogues, a écrit un article titré « Pourquoi tout le monde a besoin d'un expert en bogue »[1] et la majorité des équipes anti-bogues sur les projets *open source* ont publié au même moment des guides sur le triage qui ont aidé les débutants à s'impliquer dans la communauté. De nombreux développeurs ont débuté leur fantastique carrière dans l'*open source* en triant les bogues et ont ainsi acquis une première expérience de gestion de projet logiciel.

Il y a aussi de nos jours des outils qui peuvent vous épargner beaucoup de temps quand arrive l'abrutissant travail de triage. Du côté serveur, l'extension « stock answers » de GNOME fournit les commentaires courants et fréquemment usités afin de les ajouter aux tickets en un clic pendant que, du côté client, vous pouvez faire tourner votre propre script GreaseMonkey ou l'extension Jetpack de Matej Cepl, appelée « bugzilla-triage-scripts »[2].

1 http://tieguy.org/talks-files/LCA-2005-paper-html/index.html.
2 https://fedorahosted.org/bugzilla-triage-scripts.

Si vous êtes un musicien moyen ou médiocre mais que vous aimez tout de même la musique par-dessus tout, vous pouvez toujours y travailler en tant que journaliste. Le développement de logiciels possède également ce genre de niches qui peuvent vous donner satisfaction, au-delà de l'idée première d'écrire du code. Cela vous prendra un peu de temps pour les trouver, mais ça vaut la peine d'y consacrer vos efforts, votre expérience et vos contacts. Avec un peu de chance et de talent, cela peut même vous permettre de gagner votre vie dans le domaine qui vous intéresse personnellement… et vous éviter de finir pisse-code.

15. La voie des tests vous conduira vers la Lumière

Jonathan “Duke” Leto

Jonathan Leto, dit « Le Duc » est un développeur de logiciel, un mathématicien dont les travaux sont publiés, un ninja de Git et un passionné de cyclisme qui vit à Portland, en Oregon. C’est l’un des développeurs principaux de la machine virtuelle Parrot[1] *et le fondateur de* Leto Labs LLC[2].

Lorsque j’ai commencé à m’impliquer dans le logiciel libre et *open source*, je n’avais aucune idée de ce que pouvaient être les tests ni de leur importance. J’avais travaillé sur des projets personnels de programmation auparavant, mais la première fois que j’ai réellement travaillé sur un projet collaboratif (c’est-à-dire en faisant un peu de *commit*) c’était pour *Yacas*, acronyme de Yet Another Computer Algebra System[3].

À ce stade de mon parcours, les tests ne venaient qu’après coup. Mon méta-algorithme général était : bidouiller du code > voir si ça fonctionne > écrire (éventuellement) un test simple pour démontrer que ça fonctionne. Un test difficile à écrire n’était généralement jamais écrit.

1 http://fr.wikipedia.org/wiki/Parrot_(machine_virtuelle).
2 http://labs.leto.net.
3 http://fr.wikipedia.org/wiki/Yacas.

C’est le premier pas sur la voie qui mène à la Lumière grâce aux tests. Vous savez que les tests sont probablement une bonne idée, mais vous n’en avez pas vu clairement les bénéfices, alors vous vous contentez de les écrire de temps en temps.

Si je pouvais ouvrir un trou de souris dans l’espace-temps et donner à mon moi plus jeune un conseil plein de sagesse sur les tests, ce serait :

> « Certains tests sont plus importants, sur le long terme, que le code qu’ils testent. »

Il y a sans doute quelques personnes qui pensent en ce moment même que je mets mon casque de protection psychique[1] quand je m’assieds pour écrire du code. Comment les tests pourraient-ils être plus importants que le code qu’ils testent ? Les tests sont la preuve que votre code marche réellement ; ils vous montrent le chemin vers l’écriture d’un code propre et vous apportent aussi la souplesse qui vous permettra de modifier le code tout en sachant que les fonctionnalités seront toujours là. En effet, plus votre code source grossit, plus vos tests sont importants, car ils vous permettent de changer une partie dudit code en sachant que le reste fonctionnera.

Une autre raison essentielle qui justifie l’écriture de tests est la possibilité de spécifier que quelque chose est un souhait explicite et non une conséquence imprévue ou un oubli. Si vous avez un cahier des charges, vous pouvez utiliser des tests pour vérifier qu’il est respecté, ce qui est très important, voire indispensable dans certaines industries. Un test, c’est comme quand on raconte une histoire : l’histoire de votre conception du code et de la façon dont il devrait fonctionner. Soit le code change et évolue, soit il mute en code infectieux[2].

---

1 [NdT] Il s’agit d’un chapeau pour se protéger contre la manipulation à distance du cerveau.

2 Équivalent approché du terme *bitrot* qui en argot de codeur désigne ce fait quasi-universel : si un bout de code ne change pas mais que tout repose sur lui, il « pourrit ». Il y a alors habituellement très peu de chances pour qu’il fonctionne tant qu’aucune modification ne sera apportée pour l’adapter à de nouveaux logiciels ou

Très souvent, vous écrirez des tests une première fois pour ensuite remettre totalement en cause votre réalisation, voire la réécrire à partir de zéro. Les tests survivent souvent au code pour lesquels ils ont été conçus à l'origine. Par exemple, un jeu de tests peut être utilisé quel que soit le nombre de fois où votre code est transformé. Les tests sont en fait l'examen de passage qui vous permettra de jeter une ancienne réalisation et de dire « cette nouvelle version a une bien meilleure structure et passe notre jeu de tests ». J'ai vu cela se produire bien des fois dans les communautés Perl et Parrot, où vous pouvez souvent me voir traîner. Les tests vous permettent de changer les choses rapidement et de savoir si quelque chose est cassé. Ils sont comme des propulseurs pour les développeurs.

Les charpentiers ont un adage qui dit quelque chose comme ça :

> « Mesurer deux fois, couper une fois. »

Le code serait la coupe, le test serait la mesure.

La méthode de développement basée sur les tests fait économiser beaucoup de temps, parce qu'au lieu de vous prendre la tête à bricoler le code sans but défini, les tests précisent votre objectif.

Les tests sont aussi un très bon retour d'expérience. Chaque fois que vous faites une nouvelle passe de test, vous savez que votre code s'améliore et qu'il a une fonctionnalité de plus ou un bogue de moins.

Il est facile de se dire « je veux ajouter 50 fonctionnalités » et de passer toute la journée à bricoler le code tout en jonglant en permanence entre différents travaux. La plupart du temps, peu de choses aboutiront. La méthode de développement basée sur les tests aide à se concentrer sur la réussite d'un seul test à la fois.

Si votre code échoue devant ne serait-ce qu'un seul test, vous avez pour mission de le faire réussir. Votre cerveau se concentre

---

nouveaux matériels.

alors sur quelque chose de très spécifique, et dans la plupart des cas cela produit de meilleurs résultats que de passer constamment d'une tâche à une autre.

La plupart des informations relatives au développement basé sur les tests sont très spécifiques à un langage ou à une situation, mais ce n'est pas une obligation. Voici comment aborder l'ajout d'une nouvelle fonctionnalité ou la correction d'un bogue dans n'importe quel langage :

1. Écrivez un test qui ne donne pas de résultats, mais qui, selon vous, sera efficace quand la fonctionnalité sera implémentée ou que le bogue sera corrigé. Mode expert : pendant l'écriture du test, pensez à l'exécuter de temps en temps, même s'il n'est pas encore fini, et tentez de deviner le message d'erreur effectif que le test renverra. À force d'écrire des tests et de les faire tourner, cela devient plus facile ;

2. Bidouillez le code ;

3. Exécutez le test. S'il marche, passez au point 4, sinon retournez au point 2 ;

4. C'est fini ! Dansez le sirtaki.

Cette méthode fonctionne pour n'importe quel type de test et n'importe quel langage. Si vous ne deviez retenir qu'une seule chose de ce texte, souvenez-vous des étapes ci-dessus.

Voici maintenant quelques directives plus générales de conduite de tests qui vous serviront bien et fonctionneront dans n'importe quelle situation :

- Comprendre la différence entre ce qu'on teste et ce qu'on utilise comme un outil pour tester autre chose ;
- Les tests sont fragiles. Vous pouvez toujours écrire un test qui contrôle la validité d'un message d'erreur. Mais que se passera-t-il si le message d'erreur change ? Que se

passera-t-il quand quelqu'un traduira votre code en catalan ? Que se passera-t-il lorsque quelqu'un exécutera votre code sur un système d'exploitation dont vous n'avez jamais entendu parler ? Plus votre test est résistant, plus il aura de valeur.

Pensez à cela quand vous écrivez des tests. Vous voulez qu'ils soient résistants, c'est-à-dire que les tests, dans la plupart des cas, ne devraient avoir à changer que quand les fonctionnalités changent. Si vous devez modifier vos tests régulièrement, sans que les fonctionnalités aient changé, c'est que vous faites une erreur quelque part.

Types de tests

Bien des personnes sont perdues quand on leur parle de tests d'intégration, tests unitaires, tests d'acceptation et autres tests à toutes les sauces. Il ne faut pas trop se soucier de ces termes. Plus vous écrirez de tests, mieux vous en distinguerez les nuances et les différences entre les tests deviendront plus apparentes. Tout le monde n'a pas la même définition de ce que sont les tests, mais il est utile d'avoir des termes pour décrire les types de tests.

Tests unitaires contre tests d'intégration

Les tests unitaires et les tests d'intégration couvrent un large spectre. Les tests unitaires testent de petits segments de code et les tests d'intégration vérifient comment ces segments se combinent. Il revient à l'auteur du test de décider ce que comprend une unité, mais c'est le plus souvent au niveau d'une fonction ou d'une méthode, même si certains langages appellent ces choses différemment.

Pour rendre cela un peu plus concret, nous établirons une analogie sommaire en utilisant des fonctions. Imaginez que f(x) et g(x) soient deux fonctions qui représentent deux unités de code. Pour l'aspect concret, supposons qu'elles représentent deux fonc-

tions spécifiques du code de base de votre projet libre et *open source*.

Un test d'intégration affirme quelque chose comme la composition de la fonction, par exemple `f (g(a)) = b.` Un test d'intégration consiste à tester la façon dont plusieurs choses s'intègrent ou travaillent ensemble, plutôt que la façon dont chaque partie fonctionne individuellement. Si l'algèbre n'est pas votre truc, une autre façon de comprendre est de considérer que les tests unitaires ne testent qu'une partie de la machine à la fois, tandis que les tests d'intégration s'assurent que la plupart des parties fonctionnent à l'unisson. Un bel exemple de test d'intégration est le test de conduite d'une voiture. Vous ne vérifiez pas la pression atmosphérique, ni ne mesurez le voltage des bougies d'allumage. Vous vous assurez que le véhicule fonctionne globalement.

La plupart du temps, il est préférable d'avoir les deux. Je commence souvent avec les tests unitaires puis j'ajoute les tests d'intégration au besoin puisqu'on a besoin d'éliminer d'abord les bogues les plus basiques, puis de trouver les bogues plus subtils issus d'un emboîtement imparfait des morceaux, à l'opposé de pièces qui ne fonctionnent pas individuellement. Beaucoup de gens écrivent d'abord des tests d'intégration puis se plongent dans les tests unitaires. Le plus important n'est pas de savoir lequel vous écrirez en premier, mais d'écrire les deux types de tests.

Vers la Lumière

La méthode de développement basée sur les tests est un chemin, pas un aboutissement. Sachez apprécier le voyage et assurez-vous de faire une pause pour respirer les fleurs si vous êtes égaré.

Aide et documentation

16. Une bonne documentation change la vie des débutants

Atul Jha

Atul Jha utilise des logiciels libres depuis 2002. Il travaille en tant que spécialiste des applications au CSS Corp[1], à Chennai en Inde. Il aime parcourir les universités, rencontrer des étudiants et propager la bonne parole du logiciel libre.

En 2002, on naviguait sur Internet dans des cybercafés en raison du coût important des accès par lignes commutées. À l'époque, la messagerie instantanée de Yahoo était très populaire et j'avais pris l'habitude de discuter sur le canal *#hackers*. Il y avait quelques fous furieux là-dedans qui prétendaient qu'ils pouvaient pirater mon mot de passe. J'étais très impatient d'en savoir plus sur le piratage et de devenir l'un d'eux. Le lendemain, je suis retourné au cybercafé et j'ai tapé « comment devenir un hacker » sur le moteur de recherche Yahoo. La toute première URL dirigeait sur le livre d'Eric S. Raymond. J'étais fou de joie à l'idée d'entrer dans le cercle des initiés.

J'ai commencé à lire le livre et à ma grande surprise la définition de *hacker* était « quelqu'un qui aime résoudre les problèmes et dépasser les limites ». Il y est également écrit : « les *hackers*

1 http://www.csscorp.com/index.php.

construisent des choses, les casseurs les brisent »[1]. Hélas, je cherchais le côté obscur, celui des *crackers*, et ce livre m'a mené de l'autre côté de la force, celui des *hackers*. J'ai continué à lire le livre et à rencontrer divers termes nouveaux tels que GNU/Linux, liste de diffusion, groupe d'utilisateurs Linux, IRC, Python et bien d'autres encore.

En poursuivant mes recherches, j'ai pu trouver un groupe d'utilisateurs de Linux à Delhi et j'ai eu l'opportunité de rencontrer de vrais hackers. J'avais l'impression d'être dans un autre monde quand ils parlaient de Perl, de RMS, du noyau, des pilotes de périphériques, de compilation et d'autres choses qui me passaient bien au-dessus de la tête.

J'étais dans un autre monde. J'ai préféré rentrer à la maison et trouver une distribution Linux quelque part. J'avais bien trop peur pour leur en demander une. J'étais loin de leur niveau, un débutant totalement idiot. J'ai réussi à en obtenir une en payant 1 000 Roupies indiennes[2] à un gars qui en faisait le commerce. J'en ai essayé beaucoup mais je n'arrivais pas à faire fonctionner le son. Cette fois-là, je décidais de consulter un canal IRC depuis le cybercafé. Je trouvai *#linux-india* et lançai : « g ok1 son ». Des injonctions fusèrent alors : « pas de langage SMS » et « RTFM ». Ça m'a fait encore plus peur et j'ai mis du temps à faire la relation entre « RTFM » et « read the fucking manual »[3].

J'étais terrifié et j'ai préféré rester à l'écart d'IRC pendant quelques semaines.

Un beau jour, j'ai reçu un courriel qui annonçait une réunion mensuelle de groupes d'utilisateurs Linux. J'avais besoin de réponses à mes nombreuses questions. C'est là que j'ai rencontré Karunakar. Il m'a demandé d'apporter mon ordinateur à son

---

1 Un *hacker* sait où et comment bidouiller un programme pour effectuer des tâches autres que celles prévues par ses concepteurs alors qu'un *cracker* est un pirate informatique spécialisé dans le cassage des protections dites de sécurité.

2 [NdT] environ 13,92 €.

3 [NdT] « lis le putain de manuel » dans la langue de Molière.

bureau, où il avait l'intégralité du dépôt de Debian. C'était nouveau pour moi, mais j'étais satisfait à l'idée de pouvoir enfin écouter de la musique sur Linux. Le lendemain, j'étais dans son bureau après avoir fait le trajet avec mon ordinateur dans un bus bondé, c'était génial. En quelques heures, Debian était opérationnel sur mon système. Il m'a aussi donné quelques livres pour débutants et une liste des commandes de base.

Le lendemain, j'étais à nouveau au cybercafé, et je lisais un autre essai d'Eric S. Raymond, intitulé Comment poser les questions de manière intelligente[1]. Je continuais de fréquenter le canal #hackers sur Yahoo chat où je m'étais fait un très bon ami, Krish, qui m'a suggéré d'acheter le livre intitulé *Commandes de références sous Linux*. Après avoir passé quelque temps avec ces livres et en utilisant tldp.org (The Linux Documentation Project[2]) comme support, j'étais devenu un utilisateur débutant sous Linux. Je n'ai jamais regardé en arrière. J'ai aussi assisté à une conférence sur Linux où j'ai rencontré quelques hackers de Yahoo : écouter leurs conférences m'a beaucoup inspiré. Quelques années plus tard, j'ai eu la chance de rencontrer Richard Stallman qui est une sorte de dieu pour beaucoup de gens dans la communauté du logiciel libre.

Je dois reconnaître que la documentation d'Eric S. Raymond a changé ma vie et sûrement celle de beaucoup d'autres. Après toutes ces années passées dans la communauté du logiciel libre, je me suis rendu compte que la documentation est la clé pour amener des débutants à participer à cette extraordinaire communauté *open source*. Mon conseil à deux balles à tous les développeurs serait : s'il vous plaît, documentez votre travail, même le plus petit, car le monde est plein de débutants qui aimeraient le comprendre. Mon blog propose un large éventail de publications, qui vont des plus simples comme l'activation de la vérification

1 http://www.linux-france.org/article/these/smart-questions/smart-questions-fr.html.
2 http://tldp.org.

orthographique dans OpenOffice à celles, plus complexes, traitant de l'installation de Django dans un environnement virtuel.

17. De l’importance des bonnes manières

Rich Bowen

Rich Bowen a travaillé dans le domaine du logiciel libre et open source *pendant près de quinze ans. La majeure partie de son travail a porté sur le serveur HTTP Apache ; il a également travaillé sur* Perl[1]*, PHP et diverses applications web. Il est l’auteur de* Apache Cookbook, The Definitive Guide to Apache *et divers autres livres. Il fait également de fréquentes apparitions dans diverses conférences sur les technologies.*

J’ai commencé à travailler sur le projet de documentation du serveur HTTP Apache en septembre 2000. Du moins, c’est à ce moment-là que j’ai réalisé mon premier *commit* dans les docs. Auparavant, j’avais présenté quelques corrections par courriel, et quelqu’un d’autre les avait appliquées.

Depuis cette période, j’ai réalisé un peu plus d’un millier de modifications de la documentation du serveur HTTP Apache, ainsi qu’une poignée de modifications du code lui-même. Ceux qui s’impliquent dans les logiciels libres et *open source* le font pour tout un tas de raisons différentes. Certains tentent de se faire un nom. La plupart essaient de « gratter là où ça les démange » comme ils disent, s’efforçant d’ajouter une fonctionnalité, ou de

1 http://fr.wikipedia.org/wiki/Perl_(langage).

créer une nouvelle brique logicielle pour satisfaire un de leurs besoins.

Je me suis impliqué dans l'écriture de la documentation sur des logiciels parce que j'avais été enrôlé pour aider à l'écriture d'un livre et que la documentation existante était vraiment horrible. Donc, pour rendre le livre cohérent, j'ai dû discuter avec différentes personnes du projet afin qu'elles contribuent à donner du sens à la documentation. Lors de la rédaction de l'ouvrage, j'ai amélioré la documentation, avant tout pour me faciliter la tâche. À peu près à la même époque, Larry Wall, le créateur du langage de programmation Perl, promouvait l'idée selon laquelle les trois principales qualités d'un programmeur étaient la paresse, l'impatience et l'arrogance. Selon moi, Larry avançait des arguments fondés et avec un sens de l'humour certain. Néanmoins, une partie non négligeable de la communauté des programmeurs a pris ses paroles comme un permis de connerie.

Ce que j'ai appris au cours de mes années à la documentation *open source* c'est que les trois vertus cardinales d'un spécialiste de la documentation et, plus généralement, de l'assistance à la clientèle, sont la paresse, la patience et l'humilité. Et que la vertu qui associe et transcende ces qualités, c'est le respect.

La paresse

Nous écrivons de la documentation pour ne pas avoir à répondre aux mêmes questions tous les jours pour le restant de notre vie. Si la documentation est insuffisante, les gens auront des difficultés à utiliser le logiciel. Bien que cela puisse être la source d'une activité lucrative de consultant, il s'agit aussi du meilleur moyen de faire avorter un projet, car les gens abandonneront, frustrés, et se tourneront alors vers d'autres choses qu'ils comprendront sans y passer des heures.

Ainsi, la paresse est la première vertu d'un rédacteur de documentation. Quand un client pose une question, nous devrions

répondre à cette question en profondeur. Et même, de la façon la plus complète possible. Nous devrions alors enregistrer cette réponse pour la postérité. Nous devrions l'enrichir avec des exemples et si possible des diagrammes et des illustrations. Nous devrions nous assurer que le texte est clair, grammaticalement correct et éloquent. Nous devrions alors l'ajouter à la documentation existante dans un endroit facile à trouver et largement référencé partout où quelqu'un pourrait le chercher.

La prochaine fois que quelqu'un posera la même question, nous pourrons lui répondre avec un lien vers la réponse. Et les questions que l'on pourrait poser après l'avoir lue devraient être la source d'améliorations et d'annotations à ce qui a déjà été écrit. C'est la vraie paresse. La paresse ne signifie pas simplement se dérober au travail. Cela veut aussi dire faire le travail si bien qu'il n'aura pas à être refait.

La patience

ll existe une tendance dans le monde de la documentation technique à être impatient et querelleur. Les sources de cette impatience sont nombreuses. Certaines personnes pensent que, comme elles ont dû travailler dur pour comprendre ces choses, vous le devez aussi. Beaucoup d'entre nous dans le monde du technique sont des autodidactes et nous avons peu de patience pour ceux qui viennent après nous et veulent accéder rapidement au succès. J'aime appeler ça le comportement du « tire-toi de mon chemin ». Ce n'est pas vraiment constructif.

Si vous ne parvenez pas à être patient avec le client, alors vous ne devriez pas être impliqué dans le service clientèle. Si vous vous mettez en colère quand quelqu'un ne comprend pas, vous devriez peut-être laisser quelqu'un d'autre gérer la question.

Bien sûr, c'est très facile à dire, mais beaucoup plus difficile à faire. Si vous êtes un expert sur un sujet particulier, les gens vont inévitablement venir à vous avec leurs questions. Vous êtes obligé

d’être patient, mais comment allez-vous y parvenir ? Cela vient avec l’humilité.

L’humilité

J’ai fait de l’assistance technique, en particulier par liste de diffusion, pendant à peu près deux ans, quand j’ai commencé à suivre des conférences techniques. Ces premières années ont été très amusantes. Des idiots venaient sur une liste de diffusion et posaient des questions stupides que des milliers d’autres perdants avaient posées avant eux. Et si seulement ils avaient regardé, ils auraient trouvé toutes les réponses déjà données auparavant. Mais ils étaient trop paresseux et trop bêtes pour le faire.

Ensuite, j’ai assisté à une conférence, et j’ai constaté plusieurs choses. Tout d’abord, j’ai découvert que les personnes qui posaient ces questions étaient des êtres humains. Ils n’étaient pas simplement un bloc de texte écrit noir sur blanc, à espacement fixe. Il s’agissait d’individus. Ils avaient des enfants. Ils avaient des loisirs. Ils connaissaient beaucoup plus de choses que moi sur une grande variété de sujets. J’ai rencontré des gens brillants pour qui la technologie était un outil qui servait à accomplir des choses non techniques. Ils souhaitaient partager leurs recettes avec d’autres cuisiniers. Ils souhaitaient aider les enfants d’Afrique de l’Ouest à apprendre à lire. Ils étaient passionnés de vin et voulaient en apprendre davantage. Ils étaient, en bref, plus intelligents que moi, et mon orgueil était le seul obstacle sur la route de leur succès.

Quand je suis revenu de cette première conférence, j’ai regardé les utilisateurs de listes de diffusion sous un tout autre jour. Ce n’étaient plus des idiots posant des questions stupides, simplement des gens qui avaient besoin d’un peu de mon aide pour pouvoir accomplir une tâche. Pour la plupart, la technologie n’était pas une passion. La technologie était seulement un outil. Il était donc compréhensible qu’ils n’aient pas passé des heures à

lire les archives de la liste de diffusion de l'année passée et choisissent plutôt de reposer des questions.

Bien entendu, si un jour les aider devient agaçant, la chose la plus polie à faire est de prendre du recul et de laisser quelqu'un d'autre répondre à la question plutôt que de leur dire que ce sont des imbéciles. Mais il faut aussi se rappeler toutes les fois où j'ai eu à poser des questions stupides.

La politesse et le respect

En fin de compte, tout cela se résume à la politesse et au respect. J'ai principalement abordé la question de l'assistance technique, mais la documentation n'est rien d'autre qu'une forme statique d'assistance technique. Elle répond aux questions que vous attendez des gens et elle offre des réponses dans une forme semi-permanente à titre de référence.

Lorsque vous écrivez cette documentation, vous devriez essayer de trouver le juste équilibre entre penser que votre lecteur est un idiot et penser qu'il devrait déjà tout savoir. D'un côté, vous lui demandez de s'assurer que l'ordinateur est branché, de l'autre, vous utilisez des mots comme « simplement » et « juste » pour faire comme si chaque tâche était triviale, laissant au lecteur l'impression qu'il n'est pas suffisamment compétent.

Cela implique d'avoir beaucoup de respect et d'empathie pour votre lecteur, en essayant de vous rappeler ce que c'est que de débuter ou d'avoir un niveau intermédiaire dans l'apprentissage d'un nouveau logiciel. Cependant, les exemples de mauvaises documentations sont si répandus, qu'il ne doit pas être difficile de s'en souvenir. Il y a des chances que vous en ayez fait l'expérience au cours de la dernière semaine.

J'aurais aimé…

J'aurais aimé être moins arrogant lorsque j'ai commencé à travailler sur la documentation *open source*. Quand je relis ce que

j'ai pu écrire sur des listes de diffusion archivées publiquement, gravées à jamais dans le marbre d'Internet, j'ai honte d'avoir pu être aussi grossier.

La plus grande vertu humaine est la politesse. Toutes les autres vertus en découlent. Si vous ne pouvez pas être poli, tout ce que vous accomplirez importera peu.

18. La documentation et moi, avant et après

Anne Gentle

Anne Gentle est une rédactrice technique acharnée et la coordinatrice de la documentation communautaire à Rackspace[1] *pour* OpenStack[2]*, un projet* open source *d'informatique dans le nuage. Avant de rejoindre OpenStack, Anne travaillait en tant que consultante de publication communautaire, en donnant une direction stratégique aux rédacteurs professionnels qui veulent produire du contenu en ligne sur des wikis contenant des pages et des commentaires générés par les utilisateurs. Sa passion pour les méthodes communautaires de documentation l'a amenée à écrire un livre sur les techniques de publication collaborative pour la documentation technique intitulé* Conversation et communauté : la documentation dans le Web social[3]*. Elle s'occupe aussi bénévolement de la maintenance de la documentation pour les manuels* FLOSS[4] *qui proposent de la documentation* open source *pour les projets* open source.

Voilà une prémisse bien étrange : vider mes tripes sur ce que j'aurais voulu savoir de l'*open source* et de la documentation. Plutôt que de vous dire ce que je veux que vous sachiez sur l'*open source* et la documentation, je dois vous dire ce que j'aurais aimé que mon moi antérieur sache. Cette demande suscite

1 http://www.rackspace.com.
2 http://fr.wikipedia.org/wiki/OpenStack.
3 http://xmlpress.net/publications/conversation-community.
4 http://fr.flossmanuals.net.

un sentiment de nostalgie ou de remords voire cette horrible énigme : « à quoi pouvais-je bien penser ? ».

En ce qui me concerne, avec juste cinq ans de moins, mon moi antérieur était une trentenaire bien installée dans sa vie professionnelle. D'autres, au contraire, se souviennent qu'ils n'étaient qu'adolescents lors de leurs premières expériences *open source*. Jono Bacon[1], dans son livre L'art de la Communauté[2], raconte comment il s'est tenu devant la porte d'un appartement, le cœur battant, alors qu'il était sur le point de rencontrer quelqu'un à qui il n'avait jamais parlé que sur le réseau, par le biais d'une communauté *open source*. J'ai moi aussi fait cette expérience de la première rencontre physique de gens que j'avais découverts en ligne, mais ma première incursion dans le monde de la documentation *open source* s'est produite lorsque j'ai répondu à une demande d'aide par courriel.

Le courriel provenait d'un ancien collègue qui me demandait de l'aide pour la documentation de l'ordinateur portable XO, le projet fondateur de l'organisation *One Laptop Per Child* (un portable pour chaque enfant). J'ai réfléchi à ce que je pensais être une proposition intéressante, j'en ai parlé à mes amis et à mon époux en me demandant si ce n'était pas une bonne occasion d'expérimenter de nouvelles techniques de documentation et d'essayer une chose que je n'avais jamais faite auparavant : une documentation basée sur un wiki. Depuis cette première expérience, j'ai rejoint *OpenStack*, un projet *open source* sur une solution d'informatique dans le nuage, et je travaille à plein temps sur la documentation et le support communautaires.

Je pense immédiatement aux nombreuses contradictions que j'ai rencontrées tout au long de la route. J'ai découvert que pour chaque observation il existe des champs et contre-champs surprenants. Par exemple, la documentation est absolument indispensable pour l'aide aux utilisateurs, l'éducation, la possibilité de

1 http://www.jonobacon.org.
2 http://www.artofcommunityonline.org.

choisir ou bien la documentation promotionnelle qui amène à adopter un logiciel. Pourtant, une communauté *open source* pourra continuer d'avancer malgré le manque de documentation ou de la doc complètement défectueuse. Voici une autre collision apparente de valeurs : la documentation pourrait être une très bonne tâche pour démarrer, un point de départ pour les nouveaux volontaires, pourtant les nouveaux membres de la communauté savent si peu de choses qu'il ne leur est pas possible d'écrire ni même d'éditer de manière efficace. En outre les petits nouveaux ne sont pas bien familiers des différents publics auxquels doit servir la documentation.

Ces derniers temps on entend dire un peu partout : « les développeurs devraient écrire la doc de développement » parce qu'ils connaissent bien ce public et par conséquent cela lui serait aussi utile qu'à eux-mêmes. D'après mon expérience, un regard frais et neuf est toujours le bienvenu sur un projet et certaines personnes ont la capacité d'écrire et de partager avec d'autres ce regard frais et empathique. Mais vous n'allez sûrement pas vous mettre à créer une culture « réservée aux novices » autour de la doc parce qu'il est important que des membres essentiels de la communauté technique apportent leur contribution aux efforts de documentation, et qu'ils encouragent aussi les autres à y participer.

Une partie du vilain petit secret sur la documentation des projets *open source* est qu'il n'existe qu'une frontière pour le moins floue entre leur documentation officielle et leur documentation officieuse. Si seulement j'avais su que les efforts de documentation devraient être sans cesse renouvelés et que de nouveaux sites web pourraient apparaître là où il n'y en avait pas… Une documentation extensive n'est pas le moyen le plus efficace pour s'initier à des projets ou des logiciels, mais un parcours sinueux dans les méandres de la documentation sur le Web peut s'avérer plus instructif pour ceux qui veulent lire entre les lignes et avoir ainsi une idée de ce qui se passe dans la

communauté grâce à la documentation. Avoir beaucoup de *forks*[1] et des publics variés peut indiquer que le produit est complexe et qu'il est très suivi. Cela peut aussi signifier que la communauté n'a mis en place aucune pratique quant à la documentation de référence ou que les efforts désorganisés sont la norme.

À mes débuts, j'aurais aimé avoir la capacité de ressentir la « température conviviale » d'une communauté en ligne. Quand vous entrez dans un restaurant rempli de tables nappées de blanc, de couples qui dînent et de conversations feutrées, l'information visuelle et auditive que vous recevez détermine l'ambiance et vous donne quelques indices sur ce que vous vous apprêtez à vivre lors de votre repas. Vous pouvez tout à fait transposer ce concept de température conviviale à une communauté en ligne.

Une communauté *open source* vous donne quelques indices dans ses listes de discussion, par exemple. Une page de présentation de la liste qui commence par de nombreuses règles et conventions sur la manière de poster, indique une gouvernance très stricte. Une liste de discussion qui contient de nombreuses publications mettant l'accent sur le fait qu'il « n'y a pas de question bête » est plus accueillante pour de nouveaux rédacteurs de documentation.

J'aurais aussi aimé connaître un moyen de faire non seulement de l'audit de contenu, c'est-à-dire lister le contenu à disposition pour le projet *open source*, mais aussi de l'audit de communauté, donc lister les membres influents de la communauté *open source*, qu'il s'agisse ou non de contributeurs.

Pour terminer, une observation à propos de l'*open source* et de la documentation que j'ai pu vérifier avec plaisir, est l'idée que la rédaction de la documentation peut s'effectuer via des « sprints » – grâce à de brusques dépenses d'énergie avec un public ciblé et un but précis, pour aboutir à un ensemble documentaire de référence.

1 [NdT] Un *fork* est un nouveau logiciel créé à partir du code source d'un logiciel existant.

J’ai été très contente d’entendre lors d’une conférence à SXSW Interactive que les sprints sont tout à fait acceptables pour la collaboration en ligne, qu’il faut s’attendre à des fluctuations du niveau d’énergie de chacun et que c’est OK. La documentation des logiciels est souvent faite à l’arrache dans les moments d’accalmie d’un cycle de release[1] et ça ne pose aucun problème dans la documentation *open source* qui est basée sur la communauté. Vous pouvez adopter une approche stratégique et coordonnée et continuer tout de même de proposer des évènements de grande intensité autour de la documentation. Ce sont des moments exaltants dans l’*open source* et mon ancien moi les a pleinement ressentis. C’est une bonne chose que vous puissiez continuer à apprendre et transformer votre moi antérieur en votre moi actuel avec un paquet de conseils en poche.

1 [NdT] La release est la publication d’un logiciel.

19. Ne vous inquiétez pas, faites confiance

Shaun McCance

Shaun McCance est impliqué dans la documentation de GNOME[1] depuis 2003 en tant que rédacteur, chef de la communauté et développeur d'outils. Il a passé la plupart de ce temps à se demander comment inciter davantage de personnes à écrire une documentation de meilleure qualité, avec un certain succès à long terme. Il propose son expérience de la documentation communautaire à travers sa société de conseil, Syllogist[2].

Alors que je m'apprêtais à écrire cet article, il s'est passé quelque chose d'énorme : GNOME 3 est sorti. C'était la première version majeure de GNOME depuis neuf ans. Tout était différent et toute la documentation existante devait être réécrite. Au même moment, nous changions notre façon de la rédiger. Nous avions jeté nos vieux manuels et étions repartis sur de nouvelles bases, avec un système d'aide dynamique par sujet, en utilisant Mallard[3].

Quelques semaines avant la sortie, une partie d'entre nous s'est réunie pour élaborer la documentation. Nous avons passé nos journées à travailler, à planifier, à écrire et à réviser. Nous avons écrit des centaines de pages malgré les changements incessants

1 http://www.gnomefr.org.
2 http://syllogist.net.
3 http://sourceforge.net/projects/mallardcms.

liés aux ultimes modifications du logiciel. Nous avions des contributeurs en ligne qui proposaient de nouvelles pages et corrigeaient ce qui existait déjà. Je n'avais jamais vu notre équipe de documentation aussi productive.

À quoi avons-nous finalement abouti ? Beaucoup de facteurs sont entrés en jeu, et je pourrais écrire un livre entier sur les nuances de la documentation *open source*. Mais ce que j'ai fait de plus important a été de m'effacer et de laisser les autres faire le travail. J'ai appris à déléguer ; et à déléguer dans les règles de l'art.

Revenons huit ans en arrière. J'ai commencé à m'impliquer dans la documentation de GNOME en 2003. Je n'avais pas vraiment d'expérience en tant que rédacteur technique à cette époque. Mon emploi m'amenait à travailler sur des outils de publication. J'ai commencé à travailler sur les outils et sur le visualiseur d'aide utilisés par la documentation de GNOME. Peu de temps après, je me suis retrouvé à la rédaction de la documentation.

En ce temps-là, la majeure partie de notre documentation était entre les mains de rédacteurs techniques professionnels de chez Sun. Ils s'occupaient d'un manuel, l'écrivaient, le relisaient et l'envoyaient sur notre dépôt CVS. Après quoi nous pouvions tous le regarder, y apprendre quelque chose et lui apporter des corrections. Mais il n'existait pas d'efforts concertés pour impliquer les gens dans le processus d'écriture.

Ce n'est pas que les rédacteurs de Sun aient essayé de protéger ou de cacher quoi que ce soit. Ils étaient avant tout rédacteurs techniques. Ils connaissaient leur travail et le faisaient bien. D'autres personnes auraient pu procéder autrement pour d'autres manuels, mais eux avaient leur technique pour mener à bien leur tâche. Utiliser un groupe de collaborateurs novices, aussi enthousiastes soient-ils, pour chaque page, revient à perdre un temps inimaginable sur des détails. C'est tout simplement contre-productif.

De manière inévitable, le vent a tourné chez Sun et leurs rédacteurs techniques ont été affectés à d'autres projets. Cela nous a laissés sans nos rédacteurs les plus prolifiques, ceux qui disposaient des meilleures connaissances. Pire que cela, nous étions laissés sans communauté et personne n'était là pour ramasser les morceaux.

Il y a des idées et des processus standards dans le monde de l'entreprise. J'ai travaillé dans le monde de l'entreprise. Je ne crois pas que quiconque remette ces idées en cause. Les gens font leur travail. Ils choisissent des missions et les terminent. Ils demandent aux autres de faire une révision, mais ils n'ouvrent pas leur travail aux nouveaux venus et aux rédacteurs moins expérimentés. Les meilleurs rédacteurs écriront sans doute le plus.

Ces idées sont d'une plate évidence, mais elles échouent lamentablement dans un projet communautaire. Vous ne développerez jamais une communauté de contributeurs si vous faites tout vous-même. Dans un projet de logiciel, vous pouvez avoir des contributeurs compétents et suffisamment impliqués pour contribuer régulièrement. Dans la documentation, cela n'arrive presque jamais.

La plupart des gens qui s'essayent à la documentation ne le font pas parce qu'ils veulent être rédacteurs techniques ni même parce qu'ils aiment écrire. Ils le font parce qu'ils veulent contribuer. Et la documentation est la seule manière qu'ils trouvent accessible. Ils ne savent pas coder. Ils ne sont artistiquement pas doués. Ils ne maîtrisent pas assez une autre langue pour faire de la traduction. Mais ils savent écrire.

C'est là que les rédacteurs professionnels lèvent les yeux au ciel. Le fait que vous soyez instruit ne signifie pas que vous puissiez écrire une bonne documentation pour l'utilisateur. Il ne s'agit pas simplement de poser des mots sur le papier. Vous devez comprendre vos utilisateurs, ce qu'ils savent, ce qu'ils veulent, les endroits où ils cherchent. Vous avez besoin de savoir comment

présenter l’information de façon compréhensible et savoir où la mettre pour qu’ils puissent la trouver.

Les rédacteurs techniques vous diront que la rédaction technique n’est pas à la portée de tous. Ils ont raison. Et c’est exactement pourquoi la chose la plus importante que les rédacteurs professionnels puissent faire pour la communauté est de ne pas écrire.

La clé pour construire une communauté efficace autour de la documentation, c’est de laisser les autres prendre les décisions, faire le travail et en récolter eux-mêmes les fruits. Il ne suffit pas de leur donner du travail en continu. La seule solution pour qu’ils s’intéressent suffisamment au projet et s’y accrochent, c’est qu’ils se sentent investis personnellement. Le sentiment de faire partie intégrante d’un projet est une source puissante de motivation.

Mais si vous ne travaillez qu’avec des rédacteurs débutants et que vous leur donnez tout le travail à faire, comment pouvez-vous avoir l’assurance que la documentation ainsi créée sera de qualité ? Une participation massive mais incontrôlée n’aboutit pas à de bons résultats. Le rôle d’un rédacteur expérimenté au sein de la communauté est d’être un professeur et un mentor. Vous devez leur apprendre comment rédiger.

Commencez par impliquer les gens assez tôt dans le planning. Planifiez toujours du bas vers le haut. La planification du haut vers le bas n’incite pas à la collaboration. Il est difficile d’impliquer les gens dans la réalisation d’une vue d’ensemble de haut niveau quand tous n’ont pas la même compréhension de cette vue d’ensemble. Mais les gens sont capables de travailler sur des segments. Ils peuvent réfléchir à des sujets particuliers d’écriture, à des tâches que les gens réalisent, à des questions que les gens peuvent se poser. Ils peuvent regarder les forums de discussion et les listes de diffusion afin de voir ce que les utilisateurs demandent.

Écrivez vous-même quelques pages. Cela donne un exemple à suivre. Il faut ensuite répartir tout le reste du travail. Laissez à d'autres la responsabilité de rubriques ou de chapitres entiers. Précisez-leur clairement quelles informations ils doivent fournir, mais laissez-les écrire. C'est en forgeant qu'on devient forgeron.

Soyez constamment disponible pour les aider ou répondre aux questions. Au moins la moitié de mon temps consacré à la documentation est passée à répondre à des questions afin que les autres puissent effectuer leur travail. Quand des brouillons sont soumis, relisez-les et discutez des critiques et des corrections avec leurs auteurs. Ne vous contentez pas de corriger vous-même.

Cela vous laisse tout de même le gros du travail à faire. Les gens complètent les pièces du puzzle, mais c'est toujours vous qui les assemblez. Au fur et à mesure qu'ils acquièrent de l'expérience, les gens s'occuperont de pièces de plus en plus grandes. Encouragez-les à s'impliquer davantage. Donnez-leur davantage de travail. Faites en sorte qu'ils vous aident à aider plus de rédacteurs. La communauté fonctionnera toute seule.

Huit ans plus tard, GNOME a réussi à créer une équipe de documentation qui se gère elle-même, résout les problèmes, prend des décisions, produit une bonne documentation et accueille constamment de nouveaux contributeurs. N'importe qui peut la rejoindre et y jouer un rôle. Telle est la clé du succès pour une communauté *open source*.



Traduction

20. Mon projet m'a appris à grandir

Runa Bhattacharjee

Ces dix dernières années, Runa Bhattacharjee a travaillé à la traduction et la localisation[1] *de nombreux projets* open source *– des interfaces de bureau aux outils de systèmes d'exploitation en passant par beaucoup de choses entre les deux. Elle a la ferme conviction que les dépôts de code en amont sont les meilleurs endroits pour soumettre toutes les modifications possibles. Elle gère également un portefeuille professionnel spécialisé dans la localisation chez Red Hat. Runa traduit et maintient des traductions en Bengali (version indienne) mais elle est toujours heureuse d'aider quiconque débute dans la localisation.*

Introduction

Carburer tard dans la nuit est l'une des formes de rébellion préférées des jeunes partout dans le monde. Que ce soit pour lire un livre avec une lampe de poche sous les couvertures, regarder les rediffusions TV ou (entre autres choses) traîner sur un canal IRC et s'acharner sur un problème agaçant dans son projet *open source* favori.

1 La localisation englobe tout le processus d'adaptation d'un produit logiciel ou documentaire à une région donnée. Cela comprend la traduction dans la langue de la région mais aussi l'adaptation aux normes, à la culture et aux besoins spécifiques de cette région du monde.

Comment tout a commencé

Voici comment tout a commencé pour moi. Permettez-moi d'abord d'écrire quelques lignes sur ma personne. Lorsque j'ai été présentée au groupe d'utilisateurs de Linux de ma ville, je partageais ma vie entre des emplois et mes études pour décrocher un *master*. Très vite, j'étais devenue contributrice sur quelques projets de localisation et j'avais commencé à traduire des interfaces de bureau (principalement). Nous utilisions des éditeurs de texte personnalisés avec des systèmes intégrés pour l'écriture et les polices. Les moteurs de rendu n'étaient pas assez matures pour afficher le scénario sans erreur sur les interfaces. Mais, malgré tout, nous continuions à traduire. Je me concentrais sur la méthode de travail que j'avais créée pour mon usage. Je récupérais le contenu à traduire des personnes qui savaient comment les choses fonctionnaient, je le traduisais du mieux que je pouvais, j'ajoutais des commentaires pour aider les réviseurs à comprendre comment j'avais appréhendé le texte, je renseignais les informations requises pour les copyrights et les crédits, puis je renvoyais tout ça aux coordinateurs.

Comment je faisais

C'était avant tout une manière simple de faire les choses. Mais, surtout, c'était ma manière à moi de les faire. Je prenais le temps de planifier mon travail sur les traductions. Celles-ci étaient ensuite révisées avant de m'être retournées pour modification. De nouveau, je planifiais quand je pourrais les reprendre en fonction de mon temps libre disponible entre les études et le travail. Lorsque je me mettais finalement au boulot, je m'asseyais pour neuf à dix heures d'un coup, en général jusqu'à l'heure où blanchit la campagne, ressentant alors un grand sentiment d'accomplissement, jusqu'à la fois suivante.

Ce qui comptait

Ce que je ne savais pas, c'est que je jouais un rôle crucial sur un plan plus global. À savoir, la planification des *releases*[1]. Donc, quand j'achevais mes modestes contributions et les envoyais aux autres, je ne prenais pas en compte leur possible inutilité du fait qu'elles arrivaient trop tard pour la version en cours et trop tôt pour la suivante (qui inclurait forcément de nombreux changements qui obligeraient à se remettre au travail). Au-delà de ça, je n'avais aucune idée de l'importance que ça prenait dans le processus de release : intégration, création de paquetages, tests de l'interface, suivi et résolution des bogues.

Comment cela m'a fait grandir

Tout cela a changé radicalement quand je me suis tournée vers un rôle plus professionnel. Subitement, je faisais la même chose mais d'une manière plus structurée. J'ai appris que faire cavalier seul comme j'en avais pris l'habitude n'était pas adapté quand on devait jongler avec deux ou trois versions planifiées. Il fallait être méticuleusement synchronisé avec les feuilles de route des projets. En travaillant sur une traduction d'interface de bureau, je devais aussi vérifier que le calendrier de traduction concordait avec celui du projet principal.

Les travaux devaient idéalement commencer immédiatement après le gel de tous les messages d'origine de l'interface. Les traducteurs pouvaient alors travailler librement jusqu'à l'échéance de la période de traduction, après quoi ils pouvaient marquer la traduction comme stable dans le dépôt principal et, finalement, les paquetages pouvaient être générés. De plus, quelques distributions de systèmes d'exploitation se synchronisaient sur ce calendrier. Les traducteurs avaient donc la responsabilité supplémentaire de s'assurer que les pré-versions des systèmes d'exploitation embarquant ce bureau seraient un minimum testées afin de véri-

1 Il s'agit de la publication d'un logiciel, sa mise à la disposition du public.

fier que les traductions de l'interface avaient du sens et ne contenaient pas d'erreur.

Ce que j'aurais dû savoir

La transition ne fut pas aisée. Je fus soudain inondée par un flot d'informations que je devais gérer et par des tâches supplémentaires que je devais réaliser. Ce qui était au départ un passe-temps et plus important encore un anti-stress est devenu tout à coup une affaire sérieuse. En y repensant, je peux dire que cela m'a probablement aidée à comprendre le processus dans son intégralité étant donné que j'ai dû tout apprendre depuis le début. Ainsi armée de cette connaissance, je peux analyser des situations avec une meilleure compréhension de toutes ses dimensions. Au moment où j'ai commencé à travailler sur les projets *open source* qui m'intéressaient, il y avait beaucoup moins de professionnels qui travaillaient à plein temps dans ce domaine. La plupart des contributeurs bénévoles travaillaient ailleurs la journée et voyaient ces projets comme un moyen d'alimenter les idées créatives qui s'étiolaient dans leurs tâches quotidiennes. Donc, beaucoup de nouveaux arrivants n'étaient jamais guidés vers une manière plus professionnelle d'organiser leurs projets. Ils ont grandi pour devenir merveilleusement doués dans ce qu'ils faisaient et ont finalement compris comment ils aimeraient équilibrer leur travail avec le reste de leurs activités.

Conclusion

Aujourd'hui, j'encadre les nouveaux arrivants et l'une des premières choses que je leur fais comprendre est comment et dans quelle partie du projet leur travail aura un impact. Élaborer un modèle de travail personnel est essentiel car cela permet de se construire un environnement où il est agréable de travailler. Cependant, avoir conscience de la structure qui est affectée par le travail inculque la discipline nécessaire pour pouvoir tenir bon face aux caprices.

Ergonomie

21. Apprenez de vos utilisateurs

Guillaume Paumier

Guillaume Paumier est photographe et physicien. Il habite à Toulouse. Wikipédien depuis longtemps, il travaille actuellement pour la Wikimedia Foundation, l'organisation à but non lucratif qui héberge Wikipédia. En tant que responsable de l'ergonomie multimédia, il a notamment étudié le comportement des utilisateurs afin de créer un nouveau système d'import de fichiers pour Wikimedia Commons, la médiathèque libre associée à Wikipédia.

Vous connaissez Wikipédia, l'encyclopédie libre et gratuite que tout le monde peut modifier ? Elle a été créée en 2001 et a récemment fêté son dixième anniversaire. Bien qu'elle soit l'un des dix sites les plus visités au monde, son interface reste très « 1.0 » quand on la compare aux possibilités qu'offrent les technologies web modernes. Certains peuvent trouver ça bien : Wikipédia est un « truc sérieux » et les utilisateurs n'ont pas à être distraits par des « paillettes » dans l'interface. Pourtant, si Wikipédia a eu du mal à recruter de nouveaux contributeurs ces dernières années, c'est en partie à cause de son interface que certains considèrent comme archaïque. Ceci explique peut-être pourquoi les enquêtes sur les participants à Wikipédia ont montré à maintes reprises qu'il s'agit principalement d'une population d'hommes jeunes, attirés par la technologie, la plupart ayant une formation en informatique et en ingénierie. En dehors du fait que la connaissance

libre et les licences libres sont issues du terreau fertile du logiciel libre et *open source*, l'interface compliquée a découragé beaucoup de contributeurs éventuels.

En 2011, alors que la majorité des plates-formes de publication collaboratives en ligne (comme WordPress, Etherpad et Google Documents) offrent un éditeur graphique, même rudimentaire, Wikipédia utilise toujours, par défaut, un ancien éditeur de texte wiki qui utilise des guillemets ("") et des crochets ([[]]) pour la mise en forme. Des efforts sont en cours afin de passer à un éditeur graphique par défaut, mais ce n'est pas un défi facile à relever[1].

Mais laissons l'éditeur de côté un moment. L'interface de Wikipédia demeure assez compliquée. Et de nombreuses fonctionnalités utiles sont difficiles à découvrir. Savez-vous que Wikipédia possède un système de contrôle de versions intégré ? Et que vous pouvez voir toutes les anciennes versions d'une page ? Savez-vous que vous pouvez voir la liste de toutes les modifications effectuées par un contributeur ? Que vous pouvez créer un lien vers une version donnée d'une page ? Savez-vous que vous pouvez exporter une page en PDF ? Que vous pouvez créer un vrai livre personnalisé à partir du contenu de Wikipédia ? Et que vous pouvez le faire livrer chez vous ?

Le modèle d'implémentation

La plupart des lecteurs de Wikipédia y arrivent via des moteurs de recherche. Les statistiques montrent qu'ils passent peu de temps sur Wikipédia une fois qu'ils ont trouvé l'information qu'ils cherchaient. Un petit nombre seulement s'attarde et explore les outils que propose l'interface. Par exemple, on critique régulièrement Wikipédia sur sa qualité et sur sa fiabilité. Nombre de ces outils rarement explorés et presque cachés pourraient s'avérer bien utiles aux lecteurs pour les aider à vérifier la fiabilité de

1 NdT : en été 2013, un nouvel éditeur visuel a été déployé, permettant de réduire les contraintes liées au code wiki.

l'information, tels que les « pages de discussion » qui témoignent des discussions (passées et en cours) entre les différents contributeurs de chaque article ayant abouti à son contenu actuel.

Wikipédia et ses projets frères (tels que Wikisource[1] et Wikimedia Commons[2]) sont propulsés par le moteur de wiki MediaWiki – et sont soutenus par la *Wikimedia Foundation* ; rien que ces noms, dans leur confusion, sont un péché contre l'ergonomie. Pendant longtemps, le développement de MediaWiki a été conduit par des développeurs de logiciels. La communauté MediaWiki est forte de nombreux développeurs ; à vrai dire, la communauté est presque exclusivement composée de développeurs. Ce n'est que récemment que des designers l'ont rejointe, recrutés par la Wikimedia Foundation pour ce rôle. Il n'y a quasiment aucun designer bénévole. De ce fait, l'application a été construite et « maquettée » exclusivement par des développeurs. Par conséquent, la forme de l'interface a naturellement suivi de très près le « modèle d'implémentation », c'est-à-dire la manière dont le logiciel est implémenté dans le code et les structures de données. Le modèle d'implémentation ne correspond que rarement au « modèle utilisateur », c'est-à-dire à la manière dont l'utilisateur imagine que le logiciel fonctionne.

Il serait injuste de dire que les développeurs ne se soucient pas des utilisateurs. Quand on crée un logiciel, le but – outre le plaisir d'apprendre des choses, d'écrire du code et de résoudre des problèmes – est de le publier afin qu'il puisse être utilisé. Ceci est particulièrement vrai dans le monde du logiciel libre et *open source* où la plupart des développeurs donnent bénévolement de leur temps et de leurs connaissances. On pourrait avancer que les développeurs sont, de fait, des utilisateurs de leurs propres produits, notamment dans le monde du logiciel libre et *open source*. Après tout, ils les ont créés ou ont rejoint leurs équipes pour une bonne raison, et c'est rarement l'argent. Par conséquent,

1 http://fr.wikisource.org/wiki/Accueil.
2 http://commons.wikimedia.org/wiki/Accueil.

les développeurs de ce type de logiciels devraient être dans une position idéale pour savoir ce que veulent leurs utilisateurs.

Mais soyons honnêtes : si vous êtes en train de lire ceci, c'est que vous n'êtes pas votre utilisateur lambda.

Le point de vue du développeur

Si vous êtes développeur, il vous est particulièrement difficile de vous mettre à la place de l'utilisateur. Tout d'abord, votre connaissance du code et de l'implémentation du logiciel vous force à observer ses fonctionnalités et son interface à travers un prisme très particulier. Vous connaissez chacune des fonctionnalités de l'application que vous avez créée. Vous savez où trouver chaque menu. Si quelque chose paraît légèrement bizarre dans l'interface, il est possible que vous l'ignoriez sans le vouloir, parce que vous savez inconsciemment que c'est lié à la façon dont la fonctionnalité est implémentée.

Imaginons que vous soyez en train de créer une application qui enregistre des données sous forme de tableau (par exemple, dans une base de données). Quand vous devrez ensuite afficher ces données pour les montrer à l'utilisateur, il est très probable que vous les représentiez comme un tableau, car c'est la façon dont vous avez implémenté leur stockage. Il vous paraîtra logique d'afficher les données dans un format cohérent avec le format de stockage. Vous aurez probablement le même réflexe pour tout autre type de structure de données séquentielles : vous aurez tendance à l'afficher sous forme de séquence dans l'interface, peut-être comme une liste. Et pourtant, un autre format d'affichage aurait peut-être été plus pertinent et facile d'utilisation pour les utilisateurs, par exemple sous forme d'une série de phrases, d'un graphique ou d'une autre représentation visuelle.

Un autre défi est votre niveau d'expertise. Comme vous souhaitez que votre application soit extraordinaire, vous allez probablement vous documenter sur le sujet pour la concevoir. En

fin de compte, vous n'allez pas seulement connaître votre application sur le bout des doigts, vous allez également devenir un expert dans le domaine lui-même. Un grand nombre de vos utilisateurs n'auront pas ce niveau d'expertise – ou n'en auront pas besoin. Ils pourraient être perdus par le niveau de détail de certaines fonctionnalités ou ne pas être familiers avec des termes inconnus des profanes.

Alors, que pouvez-vous faire pour arranger cela ?

Observez les utilisateurs. Vraiment

Observer les utilisateurs à l'œuvre avec votre application est une expérience réellement révélatrice.

Bon, pour observer comment les gens utilisent votre application, vous pouvez faire appel à une société de conseil en ergonomie ; cette société va alors recruter des volontaires avec des profils différents au sein d'un vivier de plusieurs milliers de testeurs, elle va mettre au point une grille d'entretien, louer une salle dédiée aux tests d'ergonomie qui comprendront un dispositif pour enregistrer ce qui se passe sur l'écran et une caméra pointée vers le testeur, et vous serez derrière une vitre sans tain, dans une salle d'observation, à vous taper la tête contre les murs et à jurer à chaque fois que l'utilisateur fait quelque chose qui, selon vous, n'a aucun sens. Si vous avez les moyens de le faire, alors n'hésitez pas, foncez. Ce que vous y apprendrez vous permettra de complètement changer votre point de vue. Si vous n'avez pas les moyens de recourir à une procédure de test professionnelle, tout n'est pas perdu ; vous allez simplement devoir le faire par vous-même. Asseyez-vous derrière un utilisateur pendant qu'il vous montre comment il effectue ses tâches et les intègre à son mode de travail. Soyez un observateur silencieux : votre but est d'observer et de noter tout ce qui se passe. Beaucoup de choses vont vous surprendre. Une fois que l'utilisateur a terminé, vous pouvez relire vos notes et lui poser des questions afin de mieux comprendre comment il fonctionne.

Pour vous aider à conduire ces tests vous-même, il existe d'excellents ouvrages : *Don't Make Me Think: A Common Sense Approach to Web Usability*[1], écrit par Steve Krug, *About Face 3: The Essentials of Interaction Design*, d'Alan Cooper, Robert Reimann et David Cronin, et le projet OpenUs-Ability[2]. Être observé peut être un peu intimidant pour les utilisateurs, mais je parie qu'ils seront nombreux à se porter volontaires pour vous aider à améliorer votre application. Les utilisateurs qui ne peuvent pas contribuer au code sont généralement heureux de trouver d'autres façons de contribuer au logiciel libre : vous montrer comment ils utilisent le logiciel est une manière simple de le faire. Les utilisateurs sont reconnaissants du temps que vous avez donné pour développer le logiciel et veulent vous rendre la pareille.

Vous devrez garder un esprit critique et ne pas forcément accepter toutes les modifications demandées par vos utilisateurs. Écoutez attentivement leurs histoires : elles vous donneront l'occasion d'identifier des problèmes. Mais ce n'est pas parce qu'un utilisateur réclame une fonctionnalité qu'il en a absolument besoin ; peut-être que le meilleur moyen de résoudre le problème sous-jacent est de mettre en place une fonctionnalité complètement différente. Gardez du recul par rapport aux commentaires de vos utilisateurs. Mais cela, vous le saviez probablement déjà.

Et au passage, ne faites pas non plus appel à votre famille.

Je ne dis pas ça méchamment, je suis sûr que vos parents, frères et sœurs sont des gens très bien. Mais si vous créez une application comptable et que votre sœur n'a jamais tenu la moindre comptabilité, elle sera sans doute perdue. Vous perdrez plus de temps à lui expliquer ce qu'est la comptabilité en partie double qu'à tester votre logiciel. Par contre, votre mère, qui s'est acheté un appareil photo numérique l'année dernière, pourrait être

1 Traduit en français : *Je ne veux pas chercher : optimisez la navigation de vos sites et menez vos internautes à bon port.*

2 http://www.openusability.org.

un cobaye idéal si vous travaillez sur une application de gestion de photos numériques ou d'envoi sur un site de partage en ligne populaire. Pour votre application de comptabilité, il vaut mieux demander à l'un de vos collègues ou amis qui a déjà quelques notions de comptabilité.

Variez vos cobayes

Pour des raisons qui resteront éternellement mystérieuses, les gens trouveront toujours d'innombrables façons d'utiliser et de maltraiter votre application. Ils trouveront des manières de la casser que vous n'auriez même pas imaginées dans vos pires cauchemars. Certains mettront en place des processus et des méthodes de travail avec votre application qui n'ont absolument aucun sens à vos yeux. Et, de désespoir, vous vous cognerez la tête contre les murs. D'autres utiliseront votre application avec tellement d'intelligence que vous vous sentirez idiot. Essayez de rencontrer des gens qui utilisent votre application avec des objectifs différents.

Si vous ne retenez rien d'autre…

… alors retenez ceci :

- vous serez tenté de modeler l'apparence et le comportement de votre interface sur la façon dont le logiciel fonctionne en coulisse. Vos utilisateurs peuvent vous aider à éviter ce piège ;
- les utilisateurs sont des oiseaux capricieux. Ils vont casser, maltraiter et optimiser votre application à un point que vous ne pouvez pas même pas imaginer ;
- apprenez de vos utilisateurs. Améliorez votre application en fonction de ce que vous avez appris. Vous avez tout à y gagner.

22. Atteindre la Qualité sans Nom du logiciel

Federico Mena Quintero

Federico Mena Quintero est l'un des pères fondateurs du projet GNOME et fut auparavant le mainteneur de GIMP. Il travaillait aux Red Hat Advanced Development Labs durant les débuts de GNOME puis fut l'un des premiers à être recruté chez Ximian où il a principalement travaillé sur le calendrier d'Evolution. Il travaille toujours autour de GNOME pour Novell/Suse et vit au Mexique.

Lorsque j'apprenais la programmation, j'ai remarqué que je rencontrais souvent le même problème, encore et encore. J'écrivais souvent un programme qui fonctionnait relativement bien et qui était même bien structuré. Mais après quelques modifications et améliorations, je ne pouvais plus l'améliorer davantage. Soit sa complexité me surpassait, soit il était écrit de telle manière qu'il ne permettait pas d'évolution, comme une maison que vous ne pouvez pas agrandir à cause d'un toit pentu ou que vous ne pouvez pas étendre sur les côtés à cause des murs qui l'entourent.

À mesure que je progressais, j'ai appris à gérer cette complexité. Nous apprenons tous à le faire avec différents outils et techniques : l'abstraction, l'encapsulation, l'orientation objet, les techniques fonctionnelles, etc. Nous apprenons comment

diverses techniques nous permettent d'écrire des programmes plus complexes.

Cependant, le problème d'un programme trop optimisé ou bien trop intimement enchevêtré pour être modifié a persisté. Parfois, je pensais tenir une superbe conception. Mais la modifier d'une façon quelconque l'aurait « amochée » et je ne le souhaitais pas. D'autres fois, j'avais quelque chose avec tellement de parties imbriquées que je ne pouvais plus y connecter quoi que ce soit sans que tout s'effondre sous son propre poids.

Il y a quelques années, toute cette mode de la refonte du code a déferlé sans que j'y prête trop attention. Je me disais que c'était une façon de nettoyer le code, mais après ? Je sais déjà comment prendre un morceau de code et le transformer en fonction ; je sais déjà comment prendre des morceaux de code similaires et les transformer en classes dérivées. Je sais déjà écrire du code presque entièrement propre. Quel est donc le problème ?

J'ai écarté la « refactorisation »[1] en la considérant comme destinée aux programmeurs les moins expérimentés ; comme de jolies recettes de nettoyage de code mais rien qui ne puisse être découvert par soi-même.

La même chose s'est produite avec les canevas de conception. Je pensais qu'ils donnaient simplement des noms pompeux tels que Singleton et Strategy à des types de structures de tous les jours qu'on utiliserait naturellement dans un programme. Peut-être mon ego de programmeur était-il trop démesuré pour considérer ces travaux avec sérieux. C'est alors que quelque chose s'est produit.

1 [NdT] Bien que critiqué, cet anglicisme (voir http://fr.wikipedia.org/wiki/Refactorisation) semble d'un emploi courant chez les développeurs. On pourrait parler plus simplement de remaniement.

Le travail de Christopher Alexander

Il y a quelques années, mon épouse et moi avons acheté une petite maison de plain-pied et nous souhaitions l’agrandir. Nous envisagions d’avoir un enfant et avions, par conséquent, besoin de plus d’espace. Il me fallait un vrai bureau à domicile, pas une simple niche inoccupée dans laquelle ma table de travail et mes étagères de livres tiendraient à peine. En tant que cuisiniers passionnés, nous avions tous les deux besoin d’une cuisine plus spacieuse et confortable que celle de notre maison. Mon épouse avait besoin d’une pièce bien à elle.

Nous ne voulions pas payer un architecte coûteux et aucun de nous deux n’avait la moindre connaissance en matière de construction. Comment allions-nous concevoir notre maison ?

Par moments, en naviguant sur le Web, je me rappelais que j’avais déjà vu le nom d’un auteur, le titre d’un livre ou quelque chose comme ça. Il se peut que je n’y aie pas vraiment porté attention par le passé mais, d’une manière ou d’une autre, plus je vois la même chose mentionnée, plus il est probable que je finirai par m’y intéresser suffisamment pour aller voir de quoi il s’agit. « Ah ah, plusieurs personnes ont déjà mentionné ce nom ou ce livre ; je devrais peut-être y jeter un coup d’œil. »

C’est exactement ce qui s’est passé avec le nom de Christopher Alexander. J’avais lu que c’était un architecte assez spécial (de vrais bâtiments, pas de logiciels), connecté en quelque sorte au monde du logiciel par le biais de techniques orientées objet. J’ai été passionné par son travail dès que j’ai commencé à le découvrir par mes lectures.

Dans les années 1970, Christopher Alexander était mathématicien et professeur d’architecture à l’université de Californie, Berkeley. Lui et un groupe d’architectes de la même mouvance que lui ont parcouru le monde, essayant de comprendre pourquoi il existait des endroits construits par des humains (cités, villes, parcs, immeubles, maisons) où il était très agréable de vivre,

confortables, conviviaux et jolis, tandis que d'autres endroits ne l'étaient pas. Des lieux agréables étaient présents dans toutes les architectures traditionnelles du monde – européennes, africaines, asiatiques et américaines – amenant à l'idée que l'on pouvait en déduire des facteurs communs.

Alexander et son équipe ont réparti leurs découvertes au sein d'une liste de motifs architecturaux cohérents et publié trois livres : *The Timeless Way of Building*[1], où sont décrites la philosophie et la méthode nécessaires à une bonne construction ; *A Pattern Language*[2], que je décris ci-après et *The Oregon Experiment*[3] où sont détaillées la conception et la construction d'un campus universitaire grâce à leur méthode.

Une grammaire de modèles

Un modèle (ou schéma) est un problème récurrent lors de la conception et de la construction d'objets, en lien avec les contraintes qui modèlent le problème et avec une solution connectée, quasi récursivement, à d'autres modèles-pères ou modèles-fils. Par exemple, considérons le GRADIENT D'INTIMITÉ, un modèle important dans le livre (les modèles étant en capitales dans ce livre pour en faciliter l'identification, je ferai donc de même) :

1 [NdT] « L'art de la construction intemporelle » en français.
2 [NdT] « Un langage de schémas » en français.
3 [NdT] « L'expérience de L'Oregon » en français.

GRADIENT D'INTIMITÉ

Modèles-pères, et préambule : … si vous savez à peu près où vous avez l'intention de placer les ailes du bâtiment, LES AILES DE LUMIÈRE, et combien d'étages elles auront, le NOMBRE D'ÉTAGES, et où L'ENTRÉE PRINCIPALE se trouve, alors il est temps de travailler sur la disposition approximative des zones principales de chaque niveau. Dans tout bâtiment, la relation entre les zones publiques et les zones privées est de la plus haute importance.

Énoncé du problème : à moins que les volumes d'un édifice ne soient disposés dans un ordre correspondant à leur degré d'intimité, les visites des étrangers, amis, invités, clients, famille seront toujours un peu gênantes.

Explication : je ne vais pas tout citer. Prenez, par exemple, un appartement où la seule façon d'atteindre la salle de bain est de passer d'abord par la chambre à coucher. Les visites sont toujours gênantes car vous avez l'impression de devoir d'abord ranger votre chambre si vous voulez que vos visiteurs puissent utiliser les toilettes. Ou considérez des bureaux dans lesquels vous ne voulez pas qu'un espace de travail calme soit à côté de la réception, car alors celui-ci ne sera pas calme du tout – vous voulez qu'il soit davantage privé, en retrait.

Résumé de la solution : disposez les espaces de l'édifice de façon à ce qu'ils créent une suite qui commence avec l'entrée et la partie du bâtiment ouverte au public, puis conduit aux zones un peu plus privées pour finir avec les domaines les plus intimes.

Modèles-fils à consulter : ZONES COMMUNES AU CENTRE. HALL D'ENTRÉE pour les maisons ; UNE PIÈCE PROPRE À CHACUN pour les particuliers. UNE RÉCEPTION VOUS SOUHAITE LA BIENVENUE dans les bureaux avec, BUREAU SEMI-PRIVÉ à l'arrière.

Les modèles deviennent plutôt précis, ils n'imposent cependant jamais un style ou une forme déterminée au résultat. Par exemple,

il existe un modèle qui s'appelle « ÉTAGÈRES OUVERTES ». Des placards profonds vous incitent à mettre les choses les unes derrière les autres, ainsi, vous ne pouvez plus voir ni attraper les objets qui sont au fond. Ils prennent aussi beaucoup de place. Les étagères dont la profondeur permet de ne mettre qu'un seul objet restent rangées et vous savez toujours, en un seul coup d'œil, où se trouve chaque chose. Les objets que vous utilisez fréquemment ne devraient pas être derrière une porte.

Vous pouvez ainsi entrevoir l'essence même des modèles de conceptions : de bonnes recettes éprouvées qui n'imposent pas de contraintes inutiles à votre implémentation.

Les modèles ne demandent pas un style particulier ou des décorations superflues ; le livre ne vous dit pas : « appliquez ces motifs floraux sur les rampes », mais plutôt : « les pièces d'une maison devraient être orientées de telle sorte que le soleil les éclaire harmonieusement, au moment de la journée où elles sont le plus utilisées – à l'est pour les chambres le matin, à l'ouest pour le salon l'après-midi ».

J'avais acquis un exemplaire de *A Pattern Language* peu avant de commencer à agrandir notre maison. Le livre fut une révélation : c'était le moyen d'aborder la conception de notre maison et nous pouvions alors la réaliser par nous-mêmes au lieu de payer très cher une solution inadaptée. Nous étions capables de faire un plan sommaire de notre maison et ensuite d'imaginer des détails plus fins au fur et à mesure de la construction. C'est le genre de livres qui, pendant que vous le lisez, parvient à confirmer les intuitions que vous éprouviez confusément – le genre de livres qui vous fait dire en permanence : « Bien sûr, c'est exactement à ça que je pensais ».

Design Patterns, le fameux livre publié par Gamma et autres, puise directement son inspiration dans les schémas architecturaux d'Alexander. Ils voulaient faire la même chose : dresser une liste de problèmes apparaissant fréquemment lorsque l'on programme,

puis présenter de bonnes solutions qui n'imposeront pas de contraintes inutiles à votre implémentation.

Voici ce dont j'ai pris conscience en lisant *A Pattern Language* (c'est une chose précieuse que l'on retrouve à la fois dans le domaine de l'architecture et celui des logiciels) : les modèles nous fournissent un vocabulaire pour discuter de la façon dont les objets sont construits. Il est beaucoup plus pratique de dire : « Les propriétés de cet objet ont des observateurs » plutôt que : « il est possible de lui attribuer des fonctions de rappel qui sont appelées lorsque ses propriétés changent ». Ce que je pensais être uniquement des termes pompeux n'est en fait qu'une manière d'exprimer la connaissance de manière compacte.

La Qualité Sans Nom

La plus grande partie de l'explication d'Alexander sur les modèles de conception et leur philosophie se rapporte à quelque chose qu'il appelle « la qualité sans nom ». Vous connaissez des endroits qui ont cette qualité sans nom. Elle est présente dans le café où vous aimez aller lire car la lumière de l'après-midi entre avec exactement la bonne intensité. Et il y a là-bas des chaises et des tables, c'est toujours plus ou moins rempli de gens sans pour autant que vous vous sentiez oppressé. Elle est présente au coin d'un parc où un arbre ombrage un banc. Il y a peut-être un peu d'eau vive, et il importe peu qu'il pleuve ou bien que le temps soit ensoleillé, cela semble toujours être un plaisir d'y être. Pensez à une maison de *hobbit* où tout est à portée de main, où tout est confortable et où tout est fait élégamment.

Un objet ou un endroit possède la *qualité sans nom* s'il est convivial, s'il a évolué dans le temps selon sa logique propre, s'il ne recèle pas de contradiction interne, n'essaie pas d'attirer l'attention et semble réunir toutes les qualités archétypales – comme s'il avait suivi tout naturellement la voie optimale pour aboutir à sa conception. Plus important, Alexander affirme que c'est une qualité objective et non subjective, et qu'elle peut

être mesurée et comparée. Bien que cela semble être une définition floue, c'est l'état le plus abouti dans lequel Alexander a pu l'amener lors de la première phase de son travail. La vraie révélation n'apparaîtrait que plus tard.

En tant que développeurs, nous avons tous vu de beaux programmes à un moment donné. Ce sont peut-être les exemples de Programming Pearls[1], un beau livre que tout hacker devrait lire. Peut-être avez-vous vu un algorithme bien implémenté qui regorge de justesse. Vous vous souvenez peut-être d'un morceau de code très compact, très lisible, très fonctionnel et très correct. Ce logiciel possède l*a qualité sans nom*. Il était devenu évident pour moi que je devais apprendre à écrire des logiciels qui atteignent le niveau de la qualité sans nom et que la méthode d'Alexander était le bon point de départ pour y parvenir.

Le guichet de billetterie

La thèse de doctorat d'Alexander a servi de base à son livre *Notes on the Synthesis of Form*[2], paru en 1964. Il essayait de rationaliser le processus de conception en le définissant comme une progression depuis une série de conditions préalables jusqu'à un résultat final, grâce à une analyse des forces qui déterminent le design.

Permettez-moi de citer Richard Gabriel, dont je parlerai plus tard, quand il décrit l'époque où Alexander essayait de concevoir un guichet de billetterie basé sur ses idées mathématiques :

> Alexander dit (à propos de la qualité sans nom) :
>
> *Il s'agit d'un type subtil de liberté issu de contradictions internes.* (Alexander, 1979)
>
> Cette assertion fait écho aux origines de sa recherche sur la qualité. Elle commença en 1964 alors qu'il était en train

1 http://www.cs.bell-labs.com/cm/cs/pearls.
2 [NdT] Remarques sur la synthèse de la forme.

de réaliser une étude pour le *San Francisco Bay Area Rapid Transit District* (BART[1]) basée sur le travail rapporté dans *Notes on the Synthesis of Form* (Alexander 1964), lui-même basé sur sa thèse de doctorat. L'une des idées clés de ce livre était qu'avec une bonne conception, il doit y avoir une relation sous-jacente entre la structure du problème et la structure de la solution – les bonnes conceptions passent par la rédaction d'une analyse des besoins, l'analyse de leurs interactions sur les bases d'incompatibilités potentielles, produisant ainsi une décomposition hiérarchisée des différentes parties, et la reconstitution d'une structure dont

la hiérarchie structurelle est l'exact complémentaire de la hiérarchie fonctionnelle établie durant l'analyse du programme. (Alexander, 1964)

Alexander étudiait le système de forces mises en jeu dans la conception d'un guichet. Lui et son groupe avaient rédigé un cahier des charges en 390 points pour couvrir tous les cas de figure d'usage de l'édicule. Certaines spécifications concernaient des choses telles qu'être là pour obtenir des billets, pouvoir faire de la monnaie, pouvoir déplacer les personnes qui font la queue, réduire la durée d'attente pour obtenir des billets. Il a toutefois remarqué que certaines parties du système n'étaient pas concernées par ces spécifications et que le système lui-même pouvait s'enliser parce que ces autres forces – celles qui ne faisaient pas l'objet d'une spécification – agissaient pour arriver à leur propre équilibre au sein du système. Par exemple, si une personne s'arrêtait et qu'une autre s'arrêtait également pour parler avec la première, cela pouvait créer un embouteillage susceptible de mettre en échec les mécanismes mis au point pour maintenir une circulation fluide. Bien sûr, l'absence d'embouteillage faisait partie des spécifications. Mais il n'y avait rien que les concepteurs puissent faire pour l'empêcher par le biais d'un mécanisme adapté.

1 http://fr.wikipedia.org/wiki/BART.

En tant que programmeur, ça doit vous rappeler quelque chose ? Vous pouvez élaborer une conception magnifique et parfaitement rigoureuse, mais qui s'effondrera quand vous la construirez effectivement parce que des éléments que vous n'aviez pas anticipés apparaîtront alors. Ce n'est pas un échec de votre conception, mais de quelque chose d'autre ! Richard Gabriel poursuit :

> Alexander disait ceci :
>
> *Il devint alors clair que le bon fonctionnement d'un système ne dépendait pas uniquement de la satisfaction d'une série de conditions préalables. Il s'agissait plutôt d'un système qui trouve sa cohérence en lui-même et en équilibre avec les forces internes générées par ledit système, que de l'accord avec une série quelconque de conditions préalables que nous aurions arbitrairement définie. Cela m'intriguait beaucoup car l'idée qui prévalait en général à l'époque (en 1964) était que, pour l'essentiel, tout était fondé sur des objectifs. Toute mon analyse des conditions préalables tendait à converger avec le point de vue de la recherche opérationnelle qui pose qu'il faut établir des objectifs, etc. Ce qui m'ennuyait, c'est qu'une analyse correcte du guichet ne pouvait se baser uniquement sur des objectifs quelconques ; il y avait des réalités qui émergeaient du centre du système lui-même et qui, peu importe votre degré de réussite, avaient un rapport avec le fait que vous ayez créé une configuration stable au regard de ces réalités.*

Et c'est le cœur du problème : comment créer une configuration stable avec les réalités qui en émergent au fur et à mesure que vous la construisez ?

La nature de l'ordre

Bien que Christopher Alexander ait eu conscience d'avoir produit quelque chose de précieux avec ses recherches et cata-

logues de modèles, il n'était pas complètement satisfait. D'où venaient les modèles ? Pouvait-on les créer à partir de rien ou devait-on se satisfaire de ce qu'avait produit jusque-là l'architecture traditionnelle ? D'ailleurs, les modèles étaient-ils réellement nécessaires ? Comment pouvait-on mieux définir et évaluer ou mesurer cette « Qualité sans nom » ?

Alexander passa les vingt années suivantes à tenter de répondre à ces questions. En étudiant le processus réel de création par lequel des environnements bien construits avaient vu le jour, il découvrit que certains processus sont indispensables pour créer des villes ou des édifices agréables – ou toute création humaine en fait. Il en vint aux conclusions suivantes :

La nature crée des choses qui ont toutes une quinzaine de propriétés en commun (je vous expliquerai plus tard). Cela se produit uniquement par des processus naturels – physique et chimie de base – bien que nous ne sachions pas clairement pourquoi des procédés très différents produisent des résultats similaires.

On retrouve ces propriétés dans les architectures traditionnelles ou les villes qui ont simplement évolué au cours du temps. Tous les modèles décrits dans *A Pattern Language* peuvent être obtenus en suivant une méthode fondée sur ces propriétés.

Chaque propriété peut également décrire une transformation de l'espace existant.

La seule façon de réussir une bonne conception consiste à utiliser ces transformations, une à la fois.

Ceci a été publié en 2003 – 2004 en quatre tomes intitulés *The Nature of Order*[1].

1 [NdT] « La nature de l'ordre » en français.

Les quinze propriétés

Le premier tome de *La nature de l'ordre* traite de quinze propriétés qui apparaissent dans tous les systèmes naturels. Je les résumerai très brièvement (voir les références pour des illustrations et de plus amples explications) :

- des niveaux d'échelle : la gamme de tailles est équilibrée, sans changement brutal dans la taille d'objets adjacents. Les éléments ont une échelle fractale ;
- des centres forts : les différentes parties de l'espace ou de la structure sont clairement identifiables ;
- des frontières solides : les lignes délimitent les choses. Dans les systèmes vivants, les bords sont les environnements les plus productifs (par exemple, toutes les créatures qui vivent au bord de l'eau) ;
- des répétitions alternées : haut/bas, épais/fin, forme A et forme B. Les objets oscillent et alternent afin de créer un bon équilibre ;
- un espace positif : l'espace adopte une belle forme convexe et close. Ce n'est pas de l'espace excédentaire. Pensez à la manière dont les cellules d'un diagramme de Voronoï grandissent vers l'extérieur à partir d'un ensemble de points ou à la manière dont les grains d'un épi de maïs se développent à partir de petits points jusqu'à ce qu'ils touchent les grains adjacents ;
- une bonne forme : les voiles d'un bateau, la coquille d'un escargot, le bec d'un oiseau. Ils parviennent à la forme optimale qui sert leur fonction, ce qui est magnifique ;
- des symétries locales : le monde n'est pas symétrique dans son ensemble. Cependant, les petites choses tendent à être symétriques parce que, de cette manière, c'est plus facile. Votre maison n'est pas symétrique, mais chaque fenêtre l'est ;
- une profonde imbrication et de l'ambiguïté : les rues sinueuses des vieilles villes. Les axones des neurones. Il

est difficile de séparer la forme du fond, ou l'avant-plan de l'arrière-plan. Deux centres forts sont renforcés si un troisième est placé entre eux de manière à ce qu'il appartienne aux deux ;
- du contraste : vous pouvez distinguer où se termine une chose et où commence la suivante parce qu'elles ne se fondent pas l'une dans l'autre ;
- des degrés : les choses se confondent les unes les autres là où c'est nécessaire. Les concentrations dans des solutions, les congères ou les talus, les câbles supportant un pont. La manière dont la bande passante décroît alors que vous vous éloignez de l'antenne ;
- des aspérités : le monde n'est ni exempt de frottement, ni doux. Les irrégularités sont bénéfiques car elles permettent à chaque élément de s'adapter à son environnement, plutôt que d'être une copie conforme qui n'irait pas aussi bien ;
- des échos : les choses se répètent et se font écho. Elles sont uniques dans la précision de leur forme mais leurs contours généraux se répètent à l'infini ;
- du vide : parfois, vous avez un grand espace vide pour la tranquillité de la forme. Un lac, une cour, le cadre d'une image ;
- de la simplicité et du calme intrinsèque : les choses sont aussi simples que possible, sans être simplistes ;
- de l'interdépendance : chaque chose est dépendante de tout le reste. On ne peut pas séparer un poisson du bassin et des plantes aquatiques. On ne peut pas séparer une colonne de la base du bâtiment.

Les transformations qui préservent la structure

Le deuxième tome de *La nature de l'ordre* décrit comment chacune de ces propriétés définit également une transformation. Par exemple :

– Des frontières solides : vous pouvez parfois transformer quelque chose positivement en lui adjoignant une frontière. Vous installez une palissade autour d'un jardin qui sert alors d'ornement, de coupe-vent afin que des vents forts ne viennent pas endommager le jardin, mais elle existe aussi comme structure en soi. Dans une interface graphique utilisateur, des boîtes à ascenseurs sans cadre sont difficiles à distinguer de l'arrière-plan de la fenêtre (pensez à toutes ces pages web blanches dont les formulaires d'entrée de texte n'ont pas de cadre). Vous placez une corniche sur le toit d'un immeuble afin que la transition entre l'immeuble et le ciel ne soit pas abrupte.
– Des symétries locales : il est plus facile de construire de petites parties de manière symétrique : parce qu'elles sont fabriquées sur un tour, parce qu'on doit y accéder des deux côtés, parce qu'elles se plient comme un livre. Faire des choses asymétriques, seulement pour l'intérêt de la chose, demande un travail supplémentaire et il est plus difficile d'obtenir quelque chose qui fonctionne bien.
– Un espace positif : vous vous sentez trop exposé quand vous êtes au bureau ? Ajoutez une étagère à mi-hauteur à côté de vous pour délimiter votre espace mais ne vous enfermez pas complètement. Est-ce que votre interface utilisateur donne l'impression qu'il y a beaucoup d'espace restant une fois que vous avez mis les contrôles en place ? Faites plutôt en sorte que les contrôles entourent l'espace utilisable.

Chacun des points qui précèdent est une transformation qui préserve la structure. Vous effectuez un changement dans la structure existante non pas en la démolissant et en la refaisant, mais en ajustant une chose après l'autre selon ces propriétés comme transformations.

En termes de logiciel, c'est surtout à cela que consiste le « *refactoring* », autrement dit la refonte du code, quand vous

traduisez les concepts en code. Réorganiser, c'est seulement appliquer des transformations qui préservent la structure ou, comme Martin Fowler – l'auteur de *Refactoring* – l'aurait présenté, des transformations qui préservent le comportement. Vous ne changez pas ce que fait le programme ; vous changez seulement la manière dont son contenu interne est structuré, élément par élément.

En extrayant un morceau de code et en l'insérant dans une fonction nommée, vous ajoutez essentiellement une frontière solide autour du code et créez un noyau robuste. En enlevant une variable globale et en ajoutant des variables de classe, vous permettez la robustesse, car chaque instance peut maintenant avoir une valeur différente dans cette variable, comme il est nécessaire. En ayant un producteur/consommateur, ou un émetteur/récepteur, vous avez des symétries locales, des imbrications fortes et ambiguës, et une forme harmonieuse.

Richard Gabriel, l'une des principales personnalités du Common Lisp[1], a étudié comment appliquer les théories d'Alexander au logiciel (et aussi à la poésie ; le code n'est-il pas similaire à la poésie après tout ?). Il donne l'exemple suivant :

1. Imaginez que vous créez la classe AppelTelephonique. C'est un objet central implicite, qui pourrait être beaucoup plus puissant.

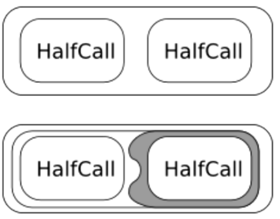


2. Gerard Meszaros dans le modèle *DemiObjet + Protocole* suggérait de séparer l'objet en deux demi-appels, liés par un protocole. On obtient ainsi une symétrie locale, un centre fort et un effet d'échelle.

3. Maintenant, dessinons cela sous forme de diagramme. On obtient alors de la symétrie locale, de l'effet d'échelle, des

1 http://fr.wikipedia.org/wiki/Common_Lisp.

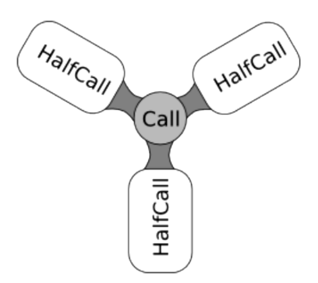
frontières, une imbrication forte et de l’ambiguïté. C’est ici que Meszaros a arrêté sa démarche.



4. Richard Gabriel suggère alors de renforcer les centres existants en appliquant d’autres transformations qui préservent la structure. Que faire de l’objet central implicite au milieu ? Vous lui ajoutez une frontière explicite (Appel) qui lie les demi-Appels entre eux. Cela améliore les symétries locales, maintient l’imbrication forte et l’ambiguïté, et c’est combinable.



5. Oui, c’est combinable. Des appels multi-directionnels, des conférences téléphoniques, tout cela s’effectue grâce à la mise en œuvre de transformations qui préservent la structure.



Chaque développeur garde probablement une image mentale du programme qu’il est en train de créer ou de modifier. La partie la plus difficile dans la modification d’un code que vous n’avez

pas écrit est de commencer par visualiser cette image mentale. Quand vous travaillez pour que le code affiche une image plus jolie, il s'améliore – et Alexander nous donne une bonne façon de le faire.

Le processus fondamental

Alexander argumente longuement pour expliquer l'intérêt de suivre ce processus : appliquer des transformations qui préservent la structure est la seule manière de réussir une conception de qualité et fonctionnelle. Cela ne vaut pas seulement pour les immeubles, mais pour tout ce que nous construisons. Peu importe que vous partiez de l'existant – un programme, un bâtiment ou une ville – ou que vous partiez de zéro. Nous imitons la nature dans ses processus d'évolution et de régénération, mais nous allons plus vite.

1. Commencez avec ce que vous avez : un espace vide, un immeuble déjà construit ou bien un programme qui ne ressemble à rien et difficile à utiliser.
2. Identifiez les centres existant dans cet espace. Trouvez le centre le plus faible ou le moins cohérent.
3. Voyez comment appliquer l'une au moins des quinze transformations qui préservent la structure afin de renforcer ce centre faible. A-t-il besoin d'être délimité ? A-t-il besoin de se confondre avec son entourage ? A-t-il besoin de plus de détails ? A-t-il besoin d'être dégagé ?
4. Trouvez les nouveaux centres qui sont apparus quand vous avez appliqué les transformations à l'ancien centre. Cette nouvelle combinaison rend-elle les choses plus fortes ? Les rend-elle plus jolies ? Les rend-elle plus fonctionnelles ?
5. Assurez-vous que vous avez fait la chose la plus simple possible.
6. Retournez au début pour l'étape suivante.

Un résumé extrêmement simple pourrait être : trouvez les mauvaises parties, améliorez-les de la façon la plus simple possible, testez les résultats, réitérez.

Alexander ne tient pas à détruire les choses juste pour les reconstruire de façon différente. Il ne s'agit pas de démolir des quartiers d'une ville pour les reconstruire mais de les améliorer progressivement. Pour les logiciels, il est bien connu que vous n'allez pas réécrire quelque chose juste parce que vous ne le comprenez plus. Démolir quelque chose, c'est perdre toutes les connaissances qui avaient été incorporées à cette chose en train d'être détruite, même si elle semble étrange dans son état actuel.

De même, Alexander s'oppose à la création de modèles détaillés au préalable. Il donne un bon argument montrant pourquoi les modèles pré-établis ne peuvent pas fonctionner en fin de compte : parce qu'on ne peut pas prévoir de manière absolue tout ce qui va se passer lors de la construction et de l'implémentation ; parce qu'on oubliera forcément une partie des détails de l'environnement au sein duquel notre création évoluera ; parce que la nature en elle-même n'est pas pré-ordonnée et croît plutôt de manière libre et pousse sans pitié à l'évolution jusqu'à ce que les éléments qui la constituent survivent d'eux-mêmes.

De cette façon, vous ne concevez pas l'interface utilisateur en entier ou la structure complète, pour un grand programme, en une seule étape. Vous allez du grand au petit ou du petit au grand (niveaux d'échelle) ; vous testez chaque partie individuellement jusqu'à ce que ce soit bon (des centres solides) ; vous vous assurez que les parties ne sont pas trop déconnectées les unes des autres (interdépendance). Vous déplacez quelques widgets là où ils sont plus accessibles ou plus proches des données auxquelles ils se réfèrent. Vous enlevez quelques cadres et séparateurs pour réduire le désordre. Par-dessus tout, vous évaluez continuellement ce que vous avez créé avec de vrais utilisateurs et des cas d'usage réels pour confronter les choses à la réalité, et non à votre imagination.

Un nom pour la qualité

Tout au long de *La nature de l'ordre*, Alexander parvient à montrer que les environnements ou les structures construits selon cette méthode finissent tous par avoir la Qualité sans Nom. Il appelle cela une structure vivante. Cela peut être mesuré et comparé. Ce n'est plus sans nom ; on peut parler d'environnements ou de programmes qui ont une structure plus ou moins vivante par rapport à d'autres – et nous tendons à développer et à obtenir toujours plus de cette propriété.

J'ai intitulé cet article « Le logiciel comme Qualité sans Nom » seulement parce que ça semblait ainsi plus mystérieux.

Je ne peux prétendre connaître la façon parfaite de concevoir et écrire des logiciels. Mais au moins, j'ai une bonne méthode basée sur ce qui produit de bonnes choses ailleurs. Cela a fonctionné pour ma maison et, jusqu'à présent, ça a très bien marché pour mes logiciels. J'espère que cela fonctionnera bien pour vous aussi !

Références

Christopher ALEXANDER, *A Pattern Language*[1].
Christopher ALEXANDER, *The Nature of Order*. Une page web très moche http://www.natureoforder.com (NdÉ : visité le 1er février 2013).
Photos et dessins des 15 propriétés - http://bit.ly/b82Dxu (NdÉ : visité le 1er février 2013).
Richard GABRIEL, *Patterns of Software*. Un superbe livre qui traite d'un grand nombre d'aspects du développement des logiciels, en transposant les idées de Christopher Alexander pour atteindre les meilleures techniques possibles en développement de

1 http://library.uniteddiversity.coop/Ecological_Building/A_Pattern_Language.pdf .

logiciel. Version en ligne : http://bit.ly/dqGUp4 (NdÉ : visité le 1er février 2013).
Richard GABRIEL, Christopher Alexander, *the search for beauty*. Une excellente présentation des idées de Christopher Alexander et une galerie de modèles dans le domaine du logiciel. http://bit.ly/ztE6cp (NdÉ : visité le 1er février 2013).
Richard GABRIEL, *The Nature of Order.* Le monde post-modèles. Une autre très bonne présentation, qui fait suite à la précédente, explique les 15 propriétés, le processus fondamental et comment cela peut s'appliquer au logiciel. http://dreamsongs.com/Files/NatureOfOrder.pdf (NdÉ : visité le 1er février 2013).
Federico MENA QUINTERO, *Software that has the Quality Without A Name*. Présentation Desktop Summit de Berlin en 2011. http://bit.ly/oYgJUf (NdÉ : visité le 1er février 2013).

Arts graphiques et design

23. Ne soyez pas timide

Máirín Duffy Strode

Máirín Duffy Strode utilise des logiciels libres et open source *depuis le lycée et y contribue depuis huit ans. Elle contribue aux communautés Fedora et GNOME et a travaillé sur l'identité visuelle des interactions, l'image de marque ou l'iconographie de plusieurs applications libres et* open source *importantes telles que Spacewalk, Anaconda, virt-manager, SELinux et SSSD. Elle s'est également engagée dans des activités de sensibilisation car elle enseigne les techniques de design à des enfants à l'aide d'outils libres et* open source *tels que GIMP et Inkscape qu'elle défend ardemment. Elle est à la tête de l'équipe de conception graphique de Fedora et* designer d'interaction[1] *senior chez Red Hat, Inc.*

Je connaissais et utilisais des logiciels libres et *open source* bien avant de devenir contributrice. Ce n'est pas faute d'avoir essayé – il y a eu quelques faux départs auxquels je n'ai pas donné suite principalement parce que j'étais trop timide et que j'ai eu peur d'aller plus loin. Sur la base de ces tentatives avortées et de ce que m'ont rapporté d'autres designers qui se sont embarqués dans des projets libres et *open source*, j'ai cinq astuces à vous offrir si vous êtes un designer aspirant au statut de contributeur au logiciel libre et *open source*.

1 http://fr.wikipedia.org/wiki/Design_d'interactiondesigner.

Sachez que l'on a besoin de vous et qu'on vous veut (très fort !)

Mon premier faux départ s'est produit alors que j'étais étudiante en première année d'informatique au Rensselaer Polytechnic Institute. Il y avait un projet particulier que j'utilisais beaucoup et auquel je voulais participer. Je ne connaissais personne au sein du projet (ni qui que ce soit investi dans le logiciel libre). J'ai donc fait une tentative à froid. Le site web du projet signalait que les contributeurs voulaient de l'aide et qu'ils avaient un canal IRC. J'y ai alors traîné pendant une semaine ou deux. Un jour, après une pause dans la conversation, j'ai osé élever la voix. J'ai dit que j'étais une étudiante en informatique intéressée par l'ergonomie et que j'adorerais participer.

On m'a répondu : « Dégage ! ». Qui plus est, on m'a fait comprendre que mon aide n'était ni nécessaire ni désirée.

Cela a retardé mon engagement de quelques années – il avait suffi de quelques mots un peu rudes sur IRC pour me dissuader de réessayer pendant près de cinq ans. Je ne découvris que bien plus tard que la personne qui m'avait plus ou moins expulsée du canal IRC de ce projet était en marge du projet, qu'elle avait un lourd passif de ce genre et que je n'avais vraiment rien fait de mal. Si seulement j'avais persévéré dans mes tentatives d'approche et conversé avec d'autres personnes, j'aurais pu commencer à ce moment-là.

Si vous souhaitez contribuer au logiciel libre et *open source*, je vous garantis qu'il y a un projet quelque part qui a vraiment besoin de votre aide, en particulier si vous êtes orienté design ! Faites-vous du design web ? De l'iconographie ? De l'ergonomie ? De l'habillage ? Des maquettes d'interface utilisateur ? J'ai parlé à de nombreux développeurs de logiciels libres et *open source* qui non seulement sont désespérément à la recherche d'aide dans ce domaine, mais qui en plus l'apprécieraient vraiment et vous vénéreraient pour l'avoir apportée.

Si vous rencontrez des résistances la première fois que vous essayez de participer dans un projet, apprenez de mon expérience et n'abandonnez pas tout de suite. Si, en définitive, le projet n'est pas fait pour vous, ne vous inquiétez pas et passez votre chemin. Il y a des chances pour que vous trouviez un projet que vous adorerez et qui vous adorera en retour.

Aidez le projet pour qu'il vous aide à aider les autres

Bien des projets libres et *open source* sont aujourd'hui dominés par les programmeurs et les ingénieurs. Et si certains ont la chance qu'une ou deux personnes créatives s'investissent, dans la plupart des projets, un designer, un artiste ou une autre présence créative représente un rêve souvent-caressé-mais-jamais-réalisé. En d'autres termes, même s'ils comprennent qu'ils ont besoin de vos compétences, ils peuvent ne pas savoir quelle sorte d'aide ils peuvent vous demander, quelle information ils doivent vous donner pour que vous puissiez être productif ni même les bases pour travailler avec vous efficacement.

Quand j'ai commencé à m'investir dans différents projets libres et *open source*, j'ai rencontré beaucoup de développeurs qui n'avaient jamais travaillé directement avec un designer auparavant. Au début, je me suis sentie un peu inutile. Je ne pouvais pas suivre toutes leurs conversations sur IRC parce qu'ils parlaient de leur cuisine interne et de détails techniques qui ne m'étaient pas familiers. Quand ils se sont donné la peine de me prêter attention, ils m'ont posé des questions comme « Quelle couleur dois-je mettre ici ? » ou « Quelle police dois-je utiliser ? ». Ce que je voulais vraiment en tant que designer d'interactions, c'était être associée à la prise de décision lorsqu'on abordait les contraintes spécifiques du projet. Si un utilisateur voulait une fonctionnalité particulière, je voulais avoir mon mot à dire sur le design – mais je ne savais pas où ni quand ces décisions se prenaient et je me sentais exclue.

Le design couvre une gamme assez large de compétences : l'illustration, la typographie, la conception des interactions, la conception visuelle, la conception d'icônes, la conception graphique, la rédaction, etc. et il y a peu de chances qu'un seul concepteur les possède toutes. Il est alors compréhensible qu'un développeur ne soit pas sûr de ce qu'il peut vous demander. Ce n'est pas qu'ils essaient de vous faire obstacle – c'est seulement qu'ils ne savent pas dans quelle mesure vous avez besoin de vous investir ou le désirez.

Aidez-les à vous aider. Montrez-leur clairement le type de contributions que vous pouvez apporter en fournissant des exemples de contributions antérieures. Faites-leur comprendre vos besoins de sorte qu'ils comprennent mieux comment vous aider à vous engager dans leur projet. Par exemple, lorsque vous vous impliquez pour la première fois dans une initiative spécifique pour le projet, prenez le temps de présenter les grandes lignes de son processus de conception et postez cela dans la liste de développement principale afin que les autres contributeurs puissent vous accompagner. Si vous avez besoin d'idées sur des points particuliers, soulignez-les dans votre présentation. Si vous n'êtes pas certain de la façon dont certaines choses se produisent – comme le processus de développement d'une nouvelle fonctionnalité – entrez en contact avec quelqu'un en parallèle et demandez-lui de vous l'expliquer pas à pas. Si quelqu'un vous demande de faire quelque chose au-delà de vos capacités techniques – travailler sur de la gestion de versions, par exemple – et que vous n'êtes pas à l'aise avec ça, dites-le.

Communiquer sur votre processus et vos besoins vous évitera de jouer aux devinettes dans le projet et ses membres seront au contraire capables d'utiliser au mieux vos talents.

Posez des questions, beaucoup de questions ; il n’y a pas de question idiote

Nous avons parfois remarqué chez Fedora que, lorsque de nouveaux designers arrivaient à bord, ils avaient peur de poser des questions techniques, par crainte de paraître stupides.

Ce qu’on ne vous dit pas, c’est que les développeurs peuvent être tellement spécialisés qu’il y a beaucoup de détails techniques qui sortent de leur domaine de compétence et qu’ils ne comprennent pas non plus – cela se produit même au sein du même projet. La différence, c’est qu’ils n’ont pas peur de demander – donc vous ne devriez pas avoir peur non plus ! Dans mon travail de design des interactions, par exemple, j’ai dû contacter de nombreuses personnes du même projet pour comprendre comment un processus se déroulait dans leur logiciel, car ce dernier comportait plusieurs sous-systèmes et tous les participants du projet ne comprenaient pas forcément comment chaque sous-système fonctionnait.

Si vous ne savez pas sur quoi travailler, que vous ne savez pas par quoi commencer ou que vous ne comprenez pas pourquoi ce que quelqu’un a dit sur le chat est si drôle – demandez. Vous avez des chances que quelqu’un vous réponde qu’il ne sait pas non plus et peu de risques de passer pour stupide. Dans la plupart des cas, vous allez apprendre quelque chose de nouveau qui vous aidera à devenir un meilleur contributeur. Il peut être particulièrement efficace de chercher un tuteur – certains projets ont même un programme de tutorat – et de lui demander s’il veut bien être votre référent quand vous avez des questions.

Partagez et partagez souvent, même si ce n’est pas encore prêt, surtout si ce n’est pas encore prêt

Nous avons aussi remarqué que de nouveaux designers pour Fedora et d’autres projets libres et *open source* sont un peu timides lorsqu’il est question de montrer leur travail. Je

comprends qu'on ne veuille pas ruiner sa réputation en publiant quelque chose qui n'est pas ce qu'on peut faire de mieux ni même fini ; mais une grande partie du travail sur des projets libres et *open source* consiste à partager souvent et ouvertement.

Plus vous aurez avancé sur un élément avant de le partager, plus il sera difficile à d'autres de vous apporter un retour utilisable, de se lancer et de s'investir. Il est aussi plus difficile pour autrui de collaborer à votre travail et d'avoir un sentiment d'appartenance au projet, de le soutenir et de le pousser jusqu'à l'implémentation. Dans certains projets libres et *open source*, ne pas être communicatif avec vos ébauches, compositions et idées est même considéré comme offensant !

Publiez vos idées, maquettes ou compositions sur le Web plutôt que par courriel, afin qu'il soit plus aisé pour les autres collaborateurs de faire référence à votre contribution en faisant un copier-coller de l'URL – c'est particulièrement pratique au cours d'une discussion. Plus vos éléments de design seront faciles à trouver, plus il est probable qu'ils seront utilisés.

Soyez aussi visible que possible au sein de la communauté du projet

Un outil qui – de manière totalement involontaire – a fini par m'aider énormément à démarrer en tant que contributeur de logiciels libres et *open source* a été mon blog. J'avais commencé à entretenir un blog, simplement pour moi, à l'image d'un portfolio grossier des choses sur lesquelles j'avais travaillé par le passé. Mon blog est un énorme atout pour moi, parce que :

- En tant qu'enregistrement de l'historique des décisions de projet, il représente un moyen pratique pour rechercher d'anciennes décisions de design – comprendre pourquoi nous avons décidé de laisser tomber tel ou tel visuel à nouveau ou pourquoi une approche particulière, précédemment essayée, n'a pas fonctionné, par exemple.

– En tant que dispositif de communication, il aide les autres contributeurs associés à votre projet et même les utilisateurs à être au courant des travaux en cours et des changements à venir pour le projet. De nombreuses fois, j'ai omis quelque chose d'essentiel dans un design et ces personnes ont très rapidement posté un commentaire pour m'en informer !

– Il m'a aidé à construire ma réputation en tant que designer de logiciels libres et *open source*, ce qui, avec le temps, m'a aidé à gagner la confiance des autres envers mes choix de design.

Vous bloguez ? Trouvez quels agrégateurs de blogs lisent les membres du projet auquel vous participez et demandez à ce que votre blog y soit ajouté (il y a en général un lien pour cela dans la barre latérale). Par exemple, l'agrégateur de blogs que vous devrez rejoindre pour faire partie de la communauté Fedora se nomme Planet Fedora[1]. Écrivez un premier billet pour vous présenter aux autres et leur faire savoir ce que vous aimez une fois que vous y aurez été ajouté – des informations du genre de celles listées dans la première astuce.

Le projet aura certainement une liste de diffusion ou un forum sur lequel les discussions ont lieu. Rejoignez-les et présentez-vous là aussi. Quand vous apportez une contribution au projet – peu importe qu'elle soit petite ou loin d'être aboutie – postez des billets sur ce que vous faites, téléchargez-la vers le wiki du projet, tweetez à ce sujet et envoyez des liens aux membres importants de la communauté via IRC afin d'avoir leur retour.

Rendez votre travail visible et les gens commenceront à vous associer à votre travail et à vous proposer des projets sympas ou d'autres opportunités, simplement grâce à ça. C'est tout ce que j'aurais aimé savoir quand j'ai essayé de m'investir pour la

1 http://planet.fedoraproject.org.

première fois dans le logiciel libre et *open source* comme designer. Si vous ne deviez retenir qu'un message de tout cela, c'est que vous ne devriez pas être timide – faites-vous entendre haut et fort, faites connaître vos besoins, faites savoir aux autres quelles sont vos capacités et ils vous aideront à les utiliser pour que le logiciel libre envoie du lourd.

24. Du bon usage des couleurs et des images dans le design

Eugene Trounev

Membre actif du logiciel libre et de KDE depuis près de six ans, Eugene Trounev a commencé chez KDEGames[1] *et s'est impliqué pendant toute la période de transition de KDE3 à KDE4. Actuellement, il s'occupe principalement de la présence de KDE sur le web et de l'apparence du bureau principal.*

Depuis la nuit des temps, les hommes ont utilisé la puissance des images et de la couleur pour transmettre des informations, attirer l'attention ou la détourner. Un célèbre dicton dit « une image vaut mille mots » et on ne saurait mieux dire. Depuis la façon dont nous nous habillons jusqu'aux néons criards des magasins de centre-ville dans le monde entier, chaque couleur, chaque forme et chaque courbe joue un rôle.

Connaître ce rôle n'est pas très difficile : toutes ces variations de teintes et de lignes sont combinées dans le but d'être perçues et ressenties par chacun de nous. Il est donc vrai qu'un design génial doit venir tout droit du cœur, puisqu'il est censé parler d'abord au cœur. Néanmoins, le cœur seul ne pourrait pas produire un design

1 http://fr.wikipedia.org/wiki/Kdegames.

génial si quelques règles n'étaient pas d'abord définies et respectées.

Des couleurs et des textures

Il existe plusieurs façons de classer les couleurs, mais la plupart mettent en avant les propriétés physiques et chimiques de la lumière ou de l'encre. Bien que cela soit important pour le résultat final, ces propriétés ne vous aideront pas à faire un design attrayant. Ce qui est selon moi le plus pertinent, c'est la division entre couleurs chaudes et couleurs froides. Pour faire simple, les couleurs chaudes sont celles qui sont les plus proches de la teinte rouge : le rouge, l'orange et le jaune. Les couleurs froides, à l'opposé du spectre, sont celles qui se rapprochent du bleu : le vert, le bleu et, dans une moindre mesure, le violet. Il est important de se rappeler que « froid » représente aussi le calme et la respiration, tandis que « chaud » est impulsif et dangereux. Ainsi, en fonction des sentiments que vous souhaitez éveiller au sein de votre public, vous devriez utiliser des couleurs plus chaudes ou plus froides. Des couleurs chaudes pour attirer l'attention ; des couleurs froides pour informer. Une utilisation excessive de l'une ou de l'autre aboutira soit à une surchauffe provoquant des sentiments négatifs chez votre public, soit à un refroidissement qui ne suscitera que son indifférence.

Il est important de se souvenir que le noir, le blanc et les nuances de gris sont aussi des couleurs. Celles-ci, toutefois, sont neutres. Elles ne provoquent aucun sentiment mais établissent plutôt une atmosphère. Leurs propriétés seront discutées plus loin.

Chaque image est d'abord et avant tout une collection de couleurs et, en tant que telle, suivra les règles de la gestion des couleurs. Déterminer la couleur dominante de votre image est la clé du succès. Essayez d'avoir une vue d'ensemble, sans vous concentrer sur les détails. Une bonne manière d'y parvenir consiste à placer une image sur un fond sombre, puis à se reculer

de quelques pas et à l'observer de loin. Quelle couleur percevez-vous le plus ?

Toutes les images n'ont toutefois pas une couleur dominante. Quelquefois, vous rencontrez un amas de couleurs dont vous ne pouvez déterminer la teinte dominante, quelle que soit l'intensité avec laquelle vous le regardez. Essayez d'éviter de telles illustrations car elles vont inévitablement déconcerter vos utilisateurs. Confrontés à des images de ce type, les gens auront tendance à rapidement regarder ailleurs et cela ne leur laissera pas une bonne impression, quel que soit le sujet abordé.

Au-delà de la couleur, les images ont aussi une texture car, en fin de compte, elles ne sont rien de plus qu'un ensemble de couleurs texturées. Identifier la texture dominante d'une image n'est pas aussi simple que de percevoir sa couleur car les textures sont rarement évidentes, particulièrement dans les photographies. Il existe néanmoins quelques indicateurs susceptibles de vous aider. La nature humaine fait que nous sommes attirés par les formes incurvées (aussi appelées formes « naturelles ») tandis que les formes anguleuses et effilées sont considérées comme moins attirantes. C'est pour cela que l'image d'une feuille verte et incurvée sera plus attirante que celle d'un pic en métal. Pour résumer : la clé d'un design réussi et séduisant est une bonne combinaison, correctement équilibrée, entre couleurs et textures au sein des images utilisées.

Textes et espaces

Un autre aspect aussi important de tout design réussi est la mise en forme du texte et la disposition des espaces autour de celui-ci. Tout comme pour les textures et les couleurs, vous devriez toujours garder à l'esprit que les gens aiment respirer. Cela signifie qu'il devrait y avoir suffisamment d'espace dans le texte et autour de lui pour le rendre plus facilement repérable, lisible et compréhensible.

Imaginez un exemple constitué de deux pages, l’une venant d’un roman d’amour, l’autre tirée d’un document à caractère juridique. Vous préféreriez très probablement le roman d’amour au document juridique. Mais savez-vous pourquoi ? La réponse est simplement que vous aimez respirer. Une page de roman d’amour contient vraisemblablement trois éléments importants : des dialogues, des paragraphes et des marges extra-larges, tandis que la plupart des documents juridiques ne comportent d’ordinaire aucun des trois. Tous les éléments mentionnés ci-dessus font vivre la page et la rendent dynamique, tandis qu’en leur absence, la page ressemble à un gros pavé de texte. L’œil humain, plus habitué à un certain degré de variation de formes, se sent plus à l’aise lorsque les pages bénéficient d’une mise en page aérée et fluide.

Toutefois, cela n’implique pas que chaque texte doive avoir ces trois éléments pour avoir l’air plus attrayant, loin de là. Tout texte peut être rendu facile et agréable à lire en l’aérant suffisamment.

Un peu de respiration ou d’espace peut être injecté au texte de plusieurs manières, telles que l’espacement des lettres, des lignes et des paragraphes ; les marges du contenu, de la section et de la page ; et enfin, la taille de la police. Essayez de garder une espace d’au moins un caractère de haut entre vos paragraphes et vos lignes ainsi que des espaces de deux caractères de haut entre vos sections. Autorisez-vous des espaces généreux autour du texte sur une page en cadrant assez largement vos marges. Essayez de ne jamais avoir une taille de police inférieure à 10 points pour vos paragraphes tout en gardant les titres assez gros pour les faire ressortir.

Attraction et information

Tout comme les animaux, les êtres humains sont souvent attirés par les éclats de couleurs brillantes et les textures inhabituelles. Plus le regard est attiré, plus les lecteurs ignoreront d’autres points d’intérêt potentiels. Cette simple règle de l’attraction est utilisée indifféremment par les hommes et les femmes pour cana-

liser l’attention des autres loin de certains éléments qui doivent selon eux passer inaperçus. Le meilleur exemple d’une telle supercherie est celui d’un magicien de rue qui distrait souvent l’attention des spectateurs par l’utilisation de fumée, de flammes ou de tenues tape-à-l’œil.

Il est important, ici, de se rappeler que les mots sont aussi visuels puisqu’ils créent des images et des associations spécifiques. La supercherie qui peut être réalisée avec de la fumée et du feu peut également être accomplie par le biais d’une utilisation créative des mots. Le meilleur exemple de supercherie quotidiennement réalisée grâce aux mots est, de loin, celle des étiquettes de prix. Vous êtes-vous déjà demandé pourquoi les commerçants aimaient tant les 99 et les 95 centimes ? C’est parce que les 9,95 €, ou même les 9,99 €, semblent plus attractifs que 10 €, même si, dans la réalité, ils ont le même impact sur votre porte-monnaie. Ajoutez-y une « vieille » étiquette de prix à 10 € ostensiblement barrée d’une épaisse ligne rouge et vous aurez un bon aimant à clients.

Conclusion

L’obtention d’un design à la fois beau et attractif passe par ces règles de base : soyez judicieux dans vos choix iconographiques ; faites un bon usage des couleurs et des textures pour créer une atmosphère ; donnez à votre lecteur des espaces pour respirer ; détournez l’attention des parties qui comptent le moins pour l’amener sur celles qui sont importantes.

Ce court article n’entend pas couvrir toute l’étendue du spectre des différents styles et techniques de design. Il s’agit plutôt de vous donner, à vous lecteur, une base sur laquelle vous pourrez vous appuyer pour construire.


Gestion de communauté

25. Comment ne pas lancer une communauté

Robert Kaye

*Robert Kaye associe ses deux passions, la musique et l'*open source*, dans l'encyclopédie de musique ouverte* MusicBrainz[1]*. Robert a créé et dirige la MetaBrainz Foundation, une organisation à but non lucratif basée en Californie, dans la continuité d'un travail de longue haleine pour améliorer l'expérience de la musique numérique. Au-delà du hack pour MusicBrainz, Robert recherche des festivals intéressants comme le* Burning Man[2] *et des projets périphériques tels que bidouiller des robots pour préparer des cocktails. Comme il est invariablement couronné d'une chevelure aux vives couleurs, vous n'aurez aucun mal à le reconnaître dans une foule.*

En 1998, je travaillais pour Xing Technology à San Luis Obispo et j'étais à fond sur notre nouveau projet AudioCatalyst. C'était l'un des premiers programmes d'extraction de MP3 intégré utilisant la base de données CDDB. Il s'agit de la base de données de CD qui permet à chaque lecteur de trouver le titre et la composition de tout CD. Si le CD n'est pas enregistré, on peut saisir les données afin que la prochaine personne qui en a besoin puisse s'en servir. J'adorais ce projet collaboratif en ligne et j'y ai enregistré des centaines de CD en quelques années.

1 http://musicbrainz.org.
2 http://www.burningman.com.

Un jour, il nous a été annoncé que CDDB avait été achetée par Escient, une société qui deviendrait GraceNote par la suite. La base de données CDDB avait été privatisée, de sorte que plus personne ne pouvait la télécharger dans son intégralité ! Pour parachever le tout, Escient ne dédommagea aucun des contributeurs pour leurs efforts. En manœuvrant ainsi, ils arnaquaient le grand public. J'étais plutôt furieux de cette décision et je le suis encore aujourd'hui.

Quelques mois plus tard, Escient nous annonçait que nous allions devoir jouer le jingle d'Escient et afficher leur logo à chaque fois que nos produits feraient une recherche de CD. Rien que ça ! J'étais vert !

Plus tard dans la même semaine, lors d'une fête avec des amis, je me plaignais de ce qui était en train de se passer et expliquais à quel point j'étais mécontent. Mon ami Kevin Murphy me demanda : « Pourquoi tu ne démarrerais pas ton propre projet *open source* pour faire concurrence à ces enfoirés ? »

Quelques semaines plus tard, je finissais de travailler pour Xing et j'avais quelques semaines de temps libre avant de commencer chez EMusic. Je décidai d'apprendre le Perl et la programmation web en autodidacte puis de démarrer la création de CD Index, un projet sans compatibilité et sans infraction avec CDDB. Je bidouillais le projet pendant cette pause mais l'ai rapidement oublié lorsque je suis devenu membre du projet FreeAmp chez EMusic.

Alors, Slashdot demanda, en mars 1999, quelle solution *open source* allait remplacer CDDB. J'ai passé le reste de la journée et la majeure partie de la nuit à finir le CD Index et à le déployer. J'ai soumis un billet sur Slashdot parlant de mon projet[1] et il fut mis en ligne rapidement. Comme prévu, des centaines de geeks se manifestèrent en quelques minutes si bien que mon serveur tomba et rendit l'âme.

1 http://slashdot.org/story/99/03/09/0923213/OpenSource-Alternative-to-CDDB.

Les masses de gens qui arrivèrent commencèrent immédiatement à se faire entendre pour que les choses se fassent. Il n'y avait même pas encore de liste de diffusion ou de logiciel de suivi de bogues ; ils insistèrent pour en avoir tout de suite. Comme j'étais novice dans l'*open source*, je ne savais pas vraiment ce qui était nécessaire ou non pour lancer un tel projet, j'ai fait comme les gens le demandaient. Les protestations reprirent de plus belle et davantage de gens encore insistèrent pour que je ferme le service étant donné qu'il n'était pas parfait. Mais même au milieu de ce vaste bazar, nous avons reçu plus de 3 000 soumissions de CD au cours des premières 24 heures.

Une fois que les choses furent calmées, il y avait encore beaucoup de gens qui rouspétaient. Greg Stein déclara qu'il allait écrire une meilleure version immédiatement. Mike Oliphant, l'auteur de Grip, annonça qu'il allait également travailler à une nouvelle version. Alan Cox vint et proclama que les bases de données SQL n'y suffiraient pas et que je devais utiliser DNS pour créer un meilleur service de recherche de CD. — Hein, quoi ? J'étais très mécontent de la communauté qui grandissait autour du billet publié sur Slashdot. Je ne voulais pas d'un lieu où les gens se manquent de respect et où certains se croient permis de gueuler encore plus fort pour obtenir ce qu'ils veulent. Je perdis rapidement tout intérêt pour le projet et CD Index déclina. Les autres projets que des personnes avaient promis de commencer (à l'exception de FreeDB) ne prirent jamais forme.

Alors, quand la bulle du point com[1] a éclaté, j'ai eu besoin de réfléchir à ce que j'allais faire ensuite. Il était clair que mon boulot chez EMusic n'avait rien de sûr ; je continuais à conduire un roadster Honda S2000, ma voiture trophée de l'époque *point com*. Avec les traites de la voiture qui doublaient mon loyer, je devais décider : soit mener ma propre entreprise et vendre ma voiture de rêve, soit déménager à Bay Area (San Francisco) et travailler sur le rêve de quelqu'un d'autre, si jamais je parvenais à

1 http://fr.wikipedia.org/wiki/Entreprise_point_com).

y trouver un travail. Je décidai que le plus intéressant serait de travailler sur une encyclopédie musicale complète construite par les utilisateurs. Je vendis la S2000 et me concentrai pour commencer à travailler sur une nouvelle mouture de CD Index.

Au cours d'une autre soirée, le nom MusicBrainz me vint et j'enregistrai le nom de domaine pendant la fête. Le jour suivant, motivé par le nouveau nom du projet, je commençai à bidouiller sérieusement et, à l'automne 2000, je lançai musicbrainz.org. Lancer n'est pas le bon mot, ici – je mis le site en place et me demandai alors comment éviter une nouvelle communauté de gosses hystériques venant de Slashdot. Je n'importai jamais de données depuis CD Index, ni ne mentionnai MusicBrainz sur les listes de diffusion de CD Index. Je me suis simplement éloigné du projet CD Index ; je ne voulais plus rien avoir à faire avec celui-ci. Finalement, j'ai décidé d'ajouter un simple bouton à la page web de FreeAmp qui mentionnait MusicBrainz.

C'est alors qu'une chose très étonnante s'est produite : des gens sont venus jeter un coup d'œil au projet. C'était seulement quelques personnes au début, mais quand quelqu'un me signalait quelque chose, je commençais une conversation et recueillais autant de retours d'informations que possible. J'améliorais le logiciel grâce à ces retours. J'ai aussi imposé un ton de respect sur les listes de discussion. À chaque fois que quelqu'un était irrespectueux, j'intervenais et haussais le ton.

Mes efforts se concentrèrent sur l'amélioration du projet. J'y ai passé plus de trois ans avant qu'il ne devienne clair que cette approche était efficace. La base de données croissait régulièrement et la qualité des données passa d'exécrable à bonne en quelques années.

Les bénévoles, ça va ça vient, mais je suis la colonne vertébrale du projet, c'est toujours moi qui donne le *la* et sa direction à l'ensemble. Aujourd'hui, nous avons une association à but non lucratif avec 325 employés dans quatre pays. Google, la BBC et

Amazon comptent parmi nos clients et notre bilan financier est bon. Je ne pense pas que cela aurait pu se produire avec la communauté CD Index.

J’aurais bien voulu savoir que les communautés ont besoin de grandir avec le temps et doivent être entretenues avec beaucoup de soin.

26. Prendre de la distance pour atteindre l'excellence

Jono Bacon

*Jono Bacon est gestionnaire de communauté, directeur technique, consultant et auteur. Il travaille actuellement comme gestionnaire de la communauté Ubuntu chez Canonical. Il dirige une équipe afin de faire croître, de stimuler et d'enthousiasmer l'ensemble de la communauté Ubuntu. Il est l'auteur d'*Art of Community[1]*, le fondateur du* Community Leadership Summit[2] *et le cofondateur du* populaire podcast LugRadio[3].

La première fois que j'ai entendu parler de Linux et d'*open source* remonte à 1998. La technologie était alors horriblement compliquée et il fallait fournir de gros efforts pour obtenir un système qui tourne correctement ; pourtant, j'ai été subjugué par le concept de communauté collaborative globale. À l'époque, je ne possédais aucune connaissance, mes compétences techniques étaient limitées et j'avais des boutons.

J'étais un adolescent typique : tourmenté, cheveux longs, tee-shirt Iron Maiden. Mon chemin était déjà tout tracé, au sens le plus traditionnel ; j'irais à l'école, puis au lycée, puis à l'université, puis je trouverais un travail.

1 http://www.artofcommunityonline.org.
2 http://www.communityleadershipsummit.com.
3 http://www.lugradio.org.

Quatorze ans plus tard, le chemin que j'ai finalement suivi n'a rien de conventionnel. Cette fascination personnelle envers le communautaire m'a conduit partout dans le monde et m'a lancé des défis captivants. Il est intéressant de prendre du recul et d'analyser cette période. Enfin, ça pourrait être intéressant pour moi… Vous préférerez peut-être passer au chapitre suivant…

… Toujours avec moi ? OK, allons-y.

La science contre l'art

J'ai toujours cru que la gestion d'une communauté était moins une science qu'un art. Je définis la science comme l'exploration de méthodes permettant de reproduire des phénomènes à travers des étapes établies et clairement comprises. Dans le monde de la science, si vous connaissez la théorie et la recette pour un résultat donné, vous pouvez souvent reproduire ce résultat comme tout un chacun.

L'art est différent. Il n'y a pas de recette pour produire une chanson populaire, pour créer une peinture exceptionnelle ou pour sculpter une statue magnifique. De même, il n'y a pas vraiment d'ensemble d'étapes reproductibles pour créer une communauté prospère. Bien sûr, il existe des astuces et des techniques pour parvenir à réunir certaines composantes du succès, mais c'est la même chose pour les autres formes d'art : nous pouvons tous apprendre les notes et les accords sur une guitare, mais cela ne veut pas dire que nous allons écrire la prochaine Bohemian Rhapsody[1]. La formule qui donne naissance à un titre comme *Bohemian Rhapsody*, c'est une dose de compétence acquise et une dose de magie.

Je ne suggère cependant pas que la gestion de communauté soit une forme d'art désespérément branchée et introvertie que seuls quelques bienheureux élus très talentueux peuvent pratiquer. Ce dont je me plains est qu'il n'y ait pas de manuel sur la façon de

1 http://fr.wikipedia.org/wiki/Bohemian_Rhapsody.

créer une communauté magnifique et enthousiasmante ; il s'agit toujours d'une dose d'apprentissage et d'une dose de magie, mais la part de magie ne vous sera pas insufflée par la grâce des dieux ; il vous faudra plutôt la chercher en explorant de nouvelles voies, en restant à l'écoute des retours et en évaluant ce qui fonctionne et ce qui ne fonctionne pas.

C'est plutôt frustrant car cela veut dire que cette « magie » ne s'obtient pas en suivant une recette unique. Mais il reste la possibilité de partager les compétences acquises, comme j'ai cherché à le faire avec The Art of Community[1] et la conférence annuelle Community Leadership Summit[2].

Avant de commencer mon introspection, et pour ceux que ma carrière n'ennuie pas mortellement, je vais résumer rapidement les communautés avec lesquelles j'ai travaillé afin de poser le contexte. En bref, j'ai commencé dans mes jeunes années chevelues par la création de l'un des premiers sites web communautaires britanniques sur Linux, appelé Linux UK, et j'ai intégré la communauté des Groupes d'utilisateurs de Linux (GUL[3]). Je me suis ensuite lancé dans la création de mon propre GUL à Wolverhampton, au Royaume-Uni, et j'ai fondé le projet Infopoint pour encourager les GUL à prêcher Linux sur les salons informatiques du Royaume-Uni. J'ai ensuite poursuivi en contribuant à la communauté KDE et en fondant le site *KDE::Enterprise* ; j'ai établi le *KDE Usability Study* et contribué de-ci, de-là à diverses petites applications. J'ai ensuite fondé le groupe d'utilisateurs PHP des West Midlands. J'ai aussi commencé à m'intéresser à GNOME. J'ai développé quelques applications (GNOME iRiver, XAMPP Control panel, Lernid, Acire) et également participé à la conception et un peu au code d'une nouvelle application audio simplifiée appelée Jokosher. C'est à peu près à cette époque que j'ai co-fondé le podcast LugRadio qui a fonctionné pendant quatre ans avec plus de deux millions de téléchargements, ce qui a

1 http://artofcommunityonline.org.
2 http://communityleadershipsummit.com.
3 http://aful.org/gul.

entraîné la mise en place de cinq évènements en direct au Royaume-Uni et aux États-Unis. À cette époque, j'ai également commencé à travailler comme consultant *open source* pour l'initiative publique OpenAdvantage[1]. Là, j'ai vraiment eu l'occasion de me faire les dents sur le communautaire en aidant des organisations à travers tous les West Midlands à passer à l'*open source*. Après quelques années passées chez OpenAdvantage, je suis parti rejoindre Canonical comme gestionnaire de la communauté Ubuntu où j'ai mis en place une équipe de quatre personnes. Ensemble, nous nous investissons dans des projets très variés chez Ubuntu et Canonical. Vous êtes toujours là ?

Ouah, vous êtes tenace. Ou assommé d'ennui. Probablement assommé. Il y aura interro à la fin, ça vous apprendra…

Réfléchir

Ceci m'amène donc à l'objet de cet article : la curieuse question de savoir ce que je me dirais si j'avais su ce que je sais aujourd'hui. À ce jour, je pense que ce que j'ai appris au cours de ma carrière peut se diviser en deux grosses catégories :

- Pratique : les trucs et astuces du métier, par exemple, les différentes approches des moyens de communication, les différentes façons d'utiliser la technologie, la gestion du temps, les différentes approches de la gestion de projet, etc.
- Personnelle : les leçons de vie et apprentissages qui modifient l'approche que nous avons de notre monde.

Je ne vais pas m'étendre sur la catégorie pratique – si vous souhaitez en savoir plus sur ce sujet, vous devriez lire mon livre (il couvre aussi l'aspect personnel). Aujourd'hui, je vais plutôt m'intéresser aux leçons de vie. Les approches et les pratiques changeront toujours, mais les leçons que l'on en tire ne changent

1 http://www.wmictcluster.org/initiatives/ict-projects/open-advantage.

pas tant que ça : elles évoluent plutôt au fur et à mesure que votre sagesse grandit.

L’importance des convictions

Les communautés sont fondamentalement des réseaux de personnes poussées par leurs convictions. Chaque communauté possède son propre état d’esprit et un domaine d’expertise propre. Cela peut être une idée aussi grandiose que de rassembler l’intégralité des savoirs de l’humanité, changer la face du monde avec le logiciel libre ou, plus simplement, animer un « club » pour que les gens puissent échanger autour de leurs livres préférés. Que ce soit pour changer le monde ou simplement pour s’amuser, chaque communauté fait valoir son propre système de valeurs ; l’humble club de lecture accorde beaucoup de valeur au fait de fournir un environnement amusant, sécurisé et libre afin de pouvoir partager des avis et des conseils de lecture. Cela ne change pas le monde, mais cela reste toujours une bonne chose à laquelle n’importe qui peut se rallier.

La règle, souvent tacite, d’une communauté est que chaque contribution d’un membre de la communauté doit bénéficier à l’ensemble de celle-ci. C’est pourquoi il est amusant d’écrire un correctif pour un bogue d’un logiciel libre, de contribuer à la documentation ou d’organiser un événement gratuit. Mais il est rare que quiconque veuille contribuer de façon bénévole si la contribution ne bénéficie qu’à une seule personne ou entreprise.

Bien sûr, je suis certain que vous tous, enfoirés cyniques que vous êtes, allez chercher et trouver une exception. Mais souvenez-vous qu’il s’agit d’une décision purement personnelle : les membres de la communauté décident par eux-mêmes si leur contribution doit bénéficier à tous. Par exemple, certains vont arguer que n’importe quelle contribution à Mono va seulement bénéficier à Microsoft et à l’ubiquité de leur infrastructure .NET. Mais des centaines de contributeurs participent à Mono parce qu’ils ne le voient pas de cette manière : ils voient leurs contribu-

tions comme un moyen utile et amusant de permettre aux développeurs d'écrire des logiciels libres plus facilement.

Si je devais parler au Jono de 1998, j'insisterais sur l'importance de cette conviction. J'en avais déjà l'intuition, mais je ne compte plus les exemples qui m'ont prouvé depuis que cette conviction incite réellement les gens à participer. J'ai souvent parlé de l'histoire du gamin d'Afrique qui m'avait envoyé un courriel pour me dire qu'il devait marcher trois heures aller-retour jusqu'au cybercafé le plus proche pour contribuer à Ubuntu. Il le faisait car il était convaincu par notre mission d'apporter le logiciel libre aux masses. On pourrait aussi citer l'énorme croissance de Wikipédia, l'incroyable rassemblement de la communauté GNOME autour de GNOME 3, le succès d'OpenStreetMap et bien d'autres exemples.

Ceci dit, la conviction n'est pas juste un artifice pour les relations publiques. Il faut qu'elle soit réelle. Bien que nous ayons tous des convictions différentes – certains ont des convictions sur le logiciel, sur l'éducation, sur le savoir, sur la transparence ou sur n'importe quoi d'autre – vous ne pouvez pas fabriquer un système de convictions à moins qu'il n'ait un but valide, auquel un groupe puisse s'intéresser. Certes, il peut être obscur, mais il doit être réel. Avec le succès du mouvement *open source*, nous avons vu des exemples d'entreprises qui tentaient de jouer cette approche et ce registre sémantique autour de la conviction, mais dans le seul but de servir leurs propres besoins. Je pourrais essayer de vous faire adhérer à cette idée : « Travaillons tous ensemble pour aider Jono à devenir riche » et fabriquer de toutes pièces quelques sophismes pour justifier cet acte de foi (par exemple, si j'étais riche, je pourrais me concentrer sur d'autres travaux, au bénéfice d'autres communautés ; mes enfants seraient élevés et éduqués dans de bien meilleures conditions, ce qui profiterait à tout le monde), mais ce serait du grand n'importe quoi.

En tant que telle, la conviction est un puissant moteur pour la contribution comme pour la collaboration, mais il est important

de l'utiliser de manière équilibrée et de l'associer au respect. Alors qu'elle peut déclencher d'incroyables changements, elle peut être terriblement dévastatrice (comme c'est le cas avec certains prédicateurs télé-évangélistes qui utilisent la religion comme un moyen de vous soustraire de l'argent, ou encore les faux médiums qui utilisent la lecture à froid[1] et s'accrochent à votre besoin désespéré de croire que vous pouvez à nouveau vous connecter à un être cher disparu).

Votre rôle

Aujourd'hui, les gestionnaires de communauté ont un rôle intéressant. Par le passé, j'avais distingué deux types de gestionnaires de communauté ; ceux qui sortent, font des conférences et s'agitent en parlant d'un produit ou d'un service et ceux qui travaillent avec une communauté de volontaires pour les aider à connaître une expérience collaborative, productive et amusante. Le second type m'intéresse plus – je pense que c'est ce que fait un vrai gestionnaire de communauté. Le premier de ces positionnements est tout à fait sympa et respectable mais convient mieux dans le domaine de la promotion et des relations publiques et nécessite d'autres compétences. J'ai quelques conseils que je pense assez intéressants pour être partagés.

La première et probablement la plus importante des leçons est d'accepter que vous puissiez avoir tort et que cela vous arrivera. Jusqu'ici, dans ma carrière, j'ai fait certaines choses correctement et j'ai commis des erreurs. Bien que je croie être généralement sur le bon chemin et que l'essentiel de mon travail soit couronné de succès, il y a eu quelques flops ici ou là. Ces loupés, accidents et faux pas n'ont jamais été dus à de la malveillance ou de la négligence ; ils ont plutôt été causés par une sous-estimation de la difficulté de mon objectif.

Ça a l'air assez évident, mais ça l'est moins quand vous avez une fonction relativement publique. En gros, les gestionnaires de

1 http://fr.wikipedia.org/wiki/Lecture_à_froid.

communautés sont souvent vus comme les représentants d'une communauté donnée. Par exemple, je sais qu'à titre personnel, on me considère comme l'une des figures publiques d'Ubuntu et cette responsabilité s'accompagne d'une certaine pression publique quant à la manière dont les gens vous perçoivent.

Avoir les projecteurs braqués sur soi en se retrouvant à la tête d'une communauté déclenche chez certains un mécanisme défensif. Ils reculent à l'idée de faire des erreurs en public, comme si les masses toujours promptes à discutailler s'attendaient à une prestation parfaite. C'est dangereux, et on a déjà vu, par le passé, des personnages publics qui ne reconnaissent jamais avoir commis une erreur par crainte d'être ridicules en public. Non seulement, c'est une idée fausse (nous faisons tous des erreurs), mais cela ne donne pas non plus à la communauté un bon exemple de chef de file honnête et transparent, tant dans les choses qu'ils font bien que dans celles qu'ils font moins bien. Il est important de se rappeler que nous éprouvons souvent du respect pour ceux qui acceptent leurs erreurs : c'est la caractéristique d'un individu honnête et accompli.

Je me rappelle mes débuts en tant que responsable chez Canonical. À l'époque, Colin Watson et Scott James Remnant, deux vieux briscards des tout débuts de Canonical et d'Ubuntu, faisaient aussi partie des responsables de l'équipe d'ingénierie d'Ubuntu. Nous avions des réunions hebdomadaires avec notre directeur technique, Matt Zimmerman, et j'y entendais régulièrement Colin et Scott avouer ouvertement qu'ils étaient mauvais dans ceci ou avaient fait une erreur dans cela ; ils étaient étonnamment humbles et acceptaient leurs forces et leurs faiblesses. En tant que responsable débutant, je l'ouvrais moins souvent, mais cela m'a appris que ce genre d'ouverture d'esprit et d'honnêteté est important non seulement en tant que responsable, mais en tant que leader d'une communauté. Depuis, je n'hésite plus à admettre publiquement mes erreurs ou à m'excuser si je foire quelque chose.

Écouter

De même qu'être ouvert aux erreurs est primordial, il est tout aussi important d'être à l'écoute et d'apprendre de ses pairs. Dans de nombreux cas, nos communautés perçoivent les responsables et les leaders de communauté comme des personnes qui devraient toujours fournir des orientations et des directions, mener activement le projet et réaliser ses objectifs. C'est sans aucun doute une responsabilité. Mais tout autant qu'être la voix qui montre la voie, il est important de prêter l'oreille pour écouter, guider quand c'est opportun et apprendre de nouvelles leçons et idées.

Les membres de nos communautés ne se réduisent pas à des mécaniques froides et insensibles qui font leur travail. Ce sont des humains qui vivent, respirent, ont des opinions, des sentiments et des idées. J'ai vu de nombreux exemples – et j'en ai déjà moi-même provoqué — de ces situations où quelqu'un est tellement habitué à donner des instructions et des conseils qu'il oublie parfois de simplement s'asseoir, écouter et apprendre de l'expérience d'un autre. Toute industrie possède ses maîtres à penser et ses experts… des gens célèbres et reconnus pour leur sagesse. Mais, d'après mon expérience, certaines des leçons de vie les plus révolutionnaires que j'ai apprises sont venues de membres tout à fait anonymes, ordinaires. Savoir écouter les gens n'est pas seulement important pour nous aider à apprendre et à nous améliorer dans ce que nous faisons, c'est également essentiel pour gagner le respect et avoir de bonnes relations avec sa communauté.

Temps de travail contre temps libre

Pendant que je parle de la façon dont nous collaborons avec notre communauté, il y a un autre résultat de mon expérience que je n'ai vraiment assimilé qu'assez récemment. Comme beaucoup de gens, de nombreux centres d'intérêt remplissent mes journées. À part être marié et essayer d'être le meilleur mari possible, et à part mon travail quotidien en tant que gestionnaire de la commu-

nauté d’Ubuntu, je participe aussi à des projets tels que Severed Fifth[1], le Community Leadership Summit et quelques autres choses. Comme on pourrait s’y attendre, je consacre mes journées à mon travail rémunéré : au boulot, je ne passe pas de temps à travailler sur ces autres projets. Et, comme on pourrait s’y attendre, lorsque ma journée de travail se termine, je me mets à travailler sur ces autres projets. La leçon qu’il faut retenir ici, c’est qu’il n’est pas toujours clair pour votre communauté de savoir où tracer les limites.

Au fil des années, j’ai développé toute une série de services en ligne que j’utilise tant dans mon travail qu’à titre personnel. Mes comptes Twitter, identi.ca et Facebook, mon blog et d’autres ressources sont les endroits où je parle de ce que je fais. Le problème est que si l’on tient compte de leur caractère public, du fait que je suis un représentant officiel du projet Ubuntu et de tous les fuseaux horaires autour du globe, même Einstein aurait du mal à faire la différence entre ce que j’écris en tant que Jono et ce que j’écris pour le compte de Canonical.

Ce qui a pu causer de la confusion. Par exemple, malgré mes clarifications répétées, OpenRespect n’est pas et n’a jamais été une initiative de Canonical. Bien sûr, quelques idiots ont choisi d’ignorer mes explications à ce sujet mais je comprends néanmoins comment la confusion a pu arriver. La même chose s’est produite pour d’autres projets tels que Severed Fifth, The Art of Community et le Community Leadership Summit, qui ne font pas et n’ont jamais fait partie de mon travail chez Canonical.

C’est une leçon pour moi car j’ai moi-même partagé un moment, cette vision des choses et disais : « Évidemment que c’est une activité de loisirs, j’ai posté ça à 20 h. » tout en haussant les épaules à propos de la confusion entre travail et projets personnels. Quand votre travail vous met dans une position relativement publique, vous ne pouvez pas vous offrir le luxe de voir les choses comme ça. Au contraire, vous devez considérer que les

1 http://www.myspace.com/severedfifth.

gens vont avoir tendance à faire la confusion et que vous allez devoir travailler plus dur pour bien clarifier les choses.

Ne voyagez pas trop

À propos du travail pour une société qui vous a recruté pour être à la tête d'une communauté, vous devriez toujours avoir conscience des risques comme des bénéfices des voyages. C'est une chose que j'ai apprise assez tôt dans ma carrière chez Canonical. Je voyais toujours les mêmes têtes dans les conférences, et il était évident que ces personnes avaient très bien communiqué sur les bénéfices des voyages auprès de leurs employeurs, tout comme je l'avais fait, sauf que j'en ai aussi appris les dangers.

Je voyageais : non seulement ce travail me fatiguait beaucoup psychologiquement, mais j'étais aussi plus loin de mes courriels, moins présent sur IRC, dans l'impossibilité d'assister à de nombreuses réunions et j'avais moins de temps à consacrer à mes engagements professionnels. Ainsi, mon rôle était principalement devenu de sortir et d'aller assister à des évènements et, même si c'était amusant, cela ne rendait pas autant service à ma communauté qu'il l'aurait fallu. Aussi ai-je radicalement réduit mes déplacements – pour tout dire, je suis allé au Linux Collab Summit il y a quelques jours, et hormis quelques évènements d'Ubuntu auxquels je devais assister, je n'ai assisté à aucune conférence depuis près d'un an. J'ai l'impression à présent d'être allé un peu trop loin dans l'autre direction. Tout est donc une question d'équilibre. Mais j'ai aussi le sentiment de mieux servir ma communauté quand je peux prendre le temps d'être au bureau, d'être en ligne et disponible.

La planification

Pour certains, être à la tête d'une communauté, ou simplement l'animer, est un rôle qui tient moins de la structure pré-établie que du fait d'être réactif. À mes débuts, c'est également ce que je pensais. Bien qu'il n'y ait absolument aucun doute sur la néces-

sité de se montrer réactif et de pouvoir répondre aux choses qui se présentent, il est également essentiel de planifier suffisamment son travail sur une période donnée.

Cette planification devrait être effectuée de manière ouverte quand c'est possible et remplit plusieurs objectifs :

– Partager la feuille de route : cela aide la communauté à comprendre sur quoi vous travaillez et lui offre souvent des occasions de vous aider.
– Apporter des garanties : cela prouve qu'un responsable de communauté fait quelque chose. Votre communauté peut constater votre travail effectif à l'œuvre. C'est très important puisque la majeure partie du travail d'un responsable de communauté s'effectue souvent sans que la communauté au sens large en ait connaissance (par exemple, avoir une conversation en tête-à-tête avec un des membres de la communauté). Et ce manque de visibilité peut parfois générer des inquiétudes sur le fait que peu de choses se produisent dans des domaines-clés alors qu'en fait, beaucoup de choses se produisent en coulisse.
– Communiquer les progrès vers le haut et vers le bas de l'organisation : c'est pertinent si vous travaillez dans une entreprise. Avoir établi de bons processus de planification montre votre travail effectif à votre hiérarchie ; ça rassure votre équipe sur le fait que vous saurez toujours sur quoi travailler et ça donne beaucoup de valeur ajoutée à la communauté.

Au fil des années, j'ai attaché de plus en plus d'importance à la planification. Mais je garde suffisamment de temps et de flexibilité pour rester réactif. Quand j'ai débuté comme gestionnaire de la communauté Ubuntu, mon planning était plutôt personnel et je le gérais au coup par coup : je prenais le pouls de la communauté et je consacrais temps et moyens à m'occuper des domaines que je jugeais adéquats.

À présent, je répartis les objectifs dans un ensemble de projets qui couvrent chacun un cycle d'Ubuntu, je rassemble des informations avec les parties prenantes, je compose une feuille de route, j'établis le suivi du travail dans des schémas directeurs. J'estime également les progrès effectués en utilisant toute une gamme d'outils et de processus tels que mon diagramme des tâches réalisées, des réunions régulières et d'autres encore. Bien que la méthode actuelle nécessite plus de planification, elle est d'une aide significative en termes de bénéfices sur les points mentionnés ci-dessus.

Perception et conflit

Dans le monde de la gestion et de la gouvernance de communautés, j'entends souvent s'exprimer l'avis selon lequel tout serait affaire de perception. En règle générale, quand j'entends ça, c'est en réponse à quelqu'un qui se trouve du mauvais côté du bâton, le plus souvent au cours d'une période de conflit.

Bien sûr, la perception joue un rôle important dans nos vies, c'est vrai ; mais ce qui peut alimenter des perceptions erronées ou inadéquates, c'est le manque d'informations, les mauvaises informations et, dans certains cas, des tensions et des tempéraments échauffés. Cela peut être une des tâches les plus complexes pour celui qui est à la tête de la communauté. Et j'en suis ressorti en tirant quelques leçons dans ce domaine également.

Les communautés sont des groupes de personnes et, dans chaque groupe, on trouve souvent des rôles stéréotypés dans lesquels les gens se glissent. On y trouve, en général, quelqu'un que l'on considère comme une rockstar ou un héros, quelqu'un de compréhensif pour les préoccupations et les soucis et qui prêtera son épaule pour pleurer, quelqu'un qui a son franc-parler et souvent quelqu'un qui est… disons… intentionnellement pénible. Les héros, les oreilles compatissantes et les grandes gueules ne posent pas particulièrement de problèmes, mais les personnes qui créent délibérément des difficultés peuvent être pénibles ;

lorsqu'il devient carrément difficile de gérer une personne, cela peut provoquer des tensions avec les autres membres et amener du conflit dans une communauté qui, sinon, est heureuse. Il faut étouffer ce genre de problèmes dans l'œuf.

Une partie du défi, ici, c'est que les gens sont des gens, que les groupes sont des groupes et qu'il n'est pas rare qu'une personne (ou un petit nombre de personnes) se fasse connaître et soit critiquée derrière son dos et passe pour quelqu'un avec qui il est difficile de travailler. De plus, la plupart des gens n'ont pas envie d'être impliqués dans un quelconque conflit et, du coup, la personne dont on se plaint ne peut pas toujours se rendre compte de la façon dont les autres la perçoivent, étant donné que personne ne veut la confronter à ce problème. Il en résulte une des situations les plus dangereuses pour les membres d'une communauté – une réputation se propage, sans même que la personne concernée ne s'en rende compte. Et du coup, puisqu'elle ne le sait pas, elle n'a jamais l'occasion de changer de comportement. C'est une situation plutôt inconfortable.

Une réponse habituelle à cette conclusion consiste à dire : « ils sont si difficiles à gérer que les arguments qu'on essaiera d'utiliser tomberont dans les oreilles d'un sourd, de toute façon ». Même si cela se produit certainement de temps à autre, ne supposez pas trop vite que ce sera la seule issue possible ; j'ai quelquefois eu la désagréable expérience de sentir que je devais faire savoir à certaines personnes la réputation qu'elles avaient développée. Et dans la plupart des cas, cela a été une réelle surprise pour elles. Et elles ont quasiment toutes modifié leur comportement après ce retour.

Sur un sujet lié, bien que ça ne soit pas si courant dans la routine quotidienne d'un gestionnaire de communauté, un conflit va souvent se pointer ici ou là. Je voudrais simplement partager deux éléments à ce propos.

La première chose, c’est de comprendre comment naissent les conflits. Pour l’expliquer, permettez-moi de vous raconter une anecdote. La semaine dernière, un de mes amis est venu en avion dans la baie de San Francisco pour une conférence. Il est arrivé le soir, je suis donc allé le chercher à l’aéroport et nous sommes allés au pub pour discuter des dernières nouvelles. Autour d’un verre, il s’est mis à m’expliquer à quel point il était déçu d’Obama et de son administration. Il a cité des exemples : la réforme de la sécurité sociale, la réforme de Wall Street, les droits du numérique et d’autres choses. Ce qui l’embêtait n’était pas la politique menée en elle-même, mais il trouvait qu’Obama n’en faisait pas assez. Ma vision des choses était légèrement différente.

Je ne suis ni un démocrate, ni un républicain ; je prends ma décision sur chaque problème, sans m’aligner sur l’une ou l’autre des parties. Là où je diffère de mon ami, cependant, c’est que je suis un peu plus favorable à Obama et son travail quotidien. C’est parce que je crois que lui, comme n’importe qui d’autre dans un poste en relation avec le public, qu’il soit internationalement reconnu comme le président ou qu’il soit obscur et dans une mission spécifique comme gestionnaire de communauté, prend conscience que l’histoire lue et comprise par le public est souvent un simple fragment de l’histoire complète. Il y a eu des cas, par le passé, où une controverse a démarré dans les communautés auxquelles j’appartenais. De nombreux commentateurs et spectateurs n’avaient pas l’entière connaissance des faits, soit parce qu’ils n’avaient pas compris les nuances et les détails du sujet, soit parce que certains éléments de l’affaire n’avaient pas été communiqués.

Bon, je sais ce que vous allez dire – quoi, certains éléments n’avaient pas été communiqués !? Il vaudrait mieux être transparent, n’est-ce pas ? Bien sûr que c’est nécessaire. Et nous devrions toujours nous attacher à être ouverts et honnêtes. Mais il y a des cas où il serait inapproprié de partager certaines parties de l’histoire. Cela pourrait être en raison de fragments de conversa-

tions privées avec des gens qui ne souhaitent pas partager leurs commentaires ou tout simplement faire son boulot bien comme il faut sans éclabousser les réputations. Par exemple, j'ai toujours eu comme pratique de ne pas faire de coups bas à des concurrents, peu importe la situation. Par le passé, il y a eu des comportements critiquables de la part de certains concurrents dans les coulisses. Mais je ne vais pas me mettre à salir leurs réputations car cela n'aurait pas vraiment d'intérêt. Je dois cependant accepter que des critiques de la communauté peuvent ne pas connaître toute la situation avec toutes les magouilles qui ont eu lieu dans les coulisses.

Enfin, à propos de conflits, je crois que j'ai appris une vraie leçon de vie au sujet de l'approche idéale des critiques et des dénouements heureux. Bien que les billets de blogs aient eu un impact très positif sur la manière dont les gens peuvent exposer leur pensée et partager leurs opinions et visions des choses, ils présentent une facette bien plus sombre. Les blogs sont aussi devenus un moyen d'exprimer parfois un peu trop rapidement des opinions excessivement enflammées. Malheureusement, j'ai un exemple plutôt embarrassant de quelqu'un qui est tombé dans ce piège : c'est votre serviteur.

Pour commencer, quelques éléments de contexte. Il existait une entreprise appelée Lindows qui distribuait une version de Linux possédant beaucoup de similarités visuelles et opérationnelles avec Windows. Microsoft désapprouva le nom « Lindows » et une bataille s'engagea pour changer le nom. D'abord, Lindows résista. Mais après que la pression avait monté, changea son nom pour s'intituler Linspire.

> Maintenant, le problème. Permettez-moi de prendre la liberté de l'expliquer dans les termes de l'article lui-même :
>
> Il y a peu de temps, un type nommé Andrew Betts a décidé de retirer les éléments non-libres de Linspire et de sortir les parties libres dans une distribution dérivée de

> Linspire qu'il a appelée Freespire. Cet acte de sortir à nouveau des distributions ou du code n'a certainement rien de nouveau et s'inscrit parfaitement dans la culture de l'*open source*. À vrai dire, bien des distributions que nous utilisons aujourd'hui ont été dérivées d'outils existants.
>
> Malheureusement, Linspire a considéré que c'était un problème et a demandé à Freespire de changer de nom. En lisant la notification de changement, le langage et le flux des phrases, j'ai l'impression que tout ça me hurle : « marketing ». Loin de moi l'idée d'insinuer que Betts a été contraint d'écrire cette page ou que les drones marketing de Linspire l'ont écrite et annexée en son nom, mais cela ne me semble pas sonner très juste. Je me serais attendu à trouver des lignes du style « Le nom de Freespire a été changé pour celui de Squiggle afin d'éviter toute confusion avec le produit Linspire. », mais ce n'est pas le cas. Au lieu de cela, on a droit à des morceaux choisis de marketing comme « Afin de prévenir toute confusion, j'ai contacté Linspire qui m'a fait une offre extrêmement généreuse pour nous tous. » La vache ! Quelle peut bien être cette offre-exclusive-qu'on-ne-rencontre-qu'une-fois-dans-sa-vie-et-qu'on-ne-trouve-pas-dans-les-magasins ? Fort heureusement, il poursuit : « Ils souhaitent que tous ceux qui ont suivi mon projet puissent faire l'expérience du vrai Linspire, ET GRATUITEMENT ! ». Maintenant, dites-nous, je vous prie, comment nous pouvons obtenir cette vraie version du logiciel « ET GRATUITEMENT » ?
>
> « Pour une période limitée, ils mettent à disposition un code de réduction dénommé FREESPIRE qui vous donnera une copie numérique gratuite de Linspire ! Veuillez visiter http://linspire.com/ pour les détails. » Ho… merci. ?

Dans un billet de mon blog, j'ai donné à Linspire un bon coup de genou dans les bijoux de famille. J'ai raconté l'histoire, protesté contre ce que je considérais comme de l'hypocrisie

considérant leur propre bataille dans des histoires similaires de marques déposées, et j'ai fulminé. J'aurais aimé que *Guitar Hero* existe à cette époque, cela aurait été un bien meilleur moyen d'utiliser mon temps.

Je me suis trompé. Mon article n'allait jamais servir à quoi que ce soit. Peu de temps après la publication de l'article, le PDG d'alors, Kevin Carmony, m'envoya un courriel. Il n'avait pas l'air satisfait du tout. Son objection, et elle était valable, visait l'absence de concertation mutuelle préalable. Ma première réaction fut de répondre sur mon blog. La réalité de l'histoire était beaucoup moins noire. Et Linspire n'était pas les ogres que j'avais dépeints. J'ai présenté mes excuses à Kevin et me suis senti idiot.

Beaucoup de conflits sont résolus par des discussions privées où les gens peuvent être ouverts et concentrés sur les solutions sans être dérangés. Au fil des années, j'ai vu beaucoup d'exemples d'une guerre publique houleuse par blogs interposés continuant, alors que, dans les coulisses, il y avait un échange calme et une recherche de solutions.

Pour conclure

Lorsque j'ai commencé à écrire cet article, il était bien plus court. Mais j'ai continué à ajouter des éléments un par un. Il est sûrement déjà assez long pour que je puisse compter le nombre de personnes lisant cette partie sur les doigts d'une main. Je vais donc l'arrêter ici. Je pourrais continuer éternellement avec des anecdotes croustillantes et des expériences dans lesquelles j'ai été suffisamment chanceux pour y être impliqué et pour étendre mes horizons. Mais je finirais par écrire *L'art de la communauté II : cette fois-ci, c'est personnel.*

La vie est une expérimentation permanente. Et j'espère que votre investissement dans la lecture de cet article vous a apporté un peu d'expérience.

27. Heureux soit l'ignorant

Alexandra Leisse

Alexandra Leisse a quitté une scène pour en rejoindre une autre. Elle a transformé son autre passion (les logiciels et le Web) en un métier. Après une période de transition de douze mois en freelance dans le logiciel et l'opéra – et noyée par de nombreuses heures d'activités dédiées à KDE, elle a rejoint Nokia et le développement de la plateforme Qt en tant que gestionnaire de la communauté web. C'est la femme derrière le réseau de développement Qt et les activités de sa communauté sur la toile. Bien qu'elle soit diplômée en art lyrique, elle refuse la plupart du temps de chanter en public.

Introduction

Quand Lydia m'a demandé de rejoindre son projet de livre sous-titré « les choses que j'aurais voulu savoir », mon esprit est resté vide. Les choses que j'aurais voulu savoir mais que je ne savais pas ? Rien ne me venait à l'esprit.

Je ne prétends pas n'avoir pas eu besoin de savoir quoi que ce soit, au contraire. J'ai eu beaucoup à apprendre et j'ai fait un nombre incalculable d'erreurs. Mais les situations ou les erreurs que j'aurais voulu éviter ? Je n'arrive pas à y penser.

Nous avons tous cette fâcheuse tendance à ne voir que les choses que nous pourrions mieux faire, celles que nous ne savons

pas, et nous les voyons comme des faiblesses. Mais si on parlait des faiblesses qui sont des atouts ?

Voici ma propre histoire sur l'ignorance, la naïveté, les mauvaises impressions, qui vous dira à quel point je suis heureuse de ne pas en avoir eu la moindre idée.

Les noms

Je n'avais aucune idée de l'identité du gars que j'ai rencontré lors de mon premier jour de travail. Il est entré dans la pièce, s'est présenté et a commencé à poser des questions me donnant l'impression que tout ce que je penserais serait insensé. Il était apparemment bien renseigné sur ce que je faisais pour KDE et les personnes que je côtoyais. Cependant, nos points de vue sur le sujet semblaient différents. À un moment, ses provocations ont fini par me fatiguer et j'ai perdu patience. Je lui ai alors dit qu'avec les personnes, ce n'était pas toujours aussi facile que les ingénieurs l'imaginent.

Juste après son départ, après une heure de discussion, je cherchais son nom sur Google : Matthias Ettrich. Ce que j'ai lu m'a expliqué pourquoi il avait posé ces questions. Si j'avais su avant qu'il était un des fondateurs du projet KDE, j'aurais débattu avec lui d'une manière bien différente, voire pas du tout.

Ces dernières années, j'ai dû chercher quelques noms et à chaque fois, j'ai été heureuse de le faire *après* le premier contact.

C'est probablement mon idée la plus importante. Lorsque j'ai rencontré toutes ces personnalités du Libre et de l'*open source* pour la première fois, je n'avais jamais entendu leurs noms auparavant. Je ne savais rien de leurs histoires, mérites ou échecs. J'ai approché tout le monde de la même façon : le contact visuel. En étant ignorante (ou naïve selon certains), je ne me sentais pas inférieure par rapport aux personnes que je rencontrais lorsque j'ai commencé mon aventure au sein du Libre et de l'*open source*. Je savais que j'avais beaucoup à apprendre mais je n'ai jamais eu

l'impression d'avoir un rang inférieur aux autres en tant qu'individu.

« Projet de grande envergure »

Je n'avais pas suivi religieusement dot.kde.org ni PlanetKDE et encore moins ces innombrables publications liées au Libre et à l'*open source*, avant de commencer à m'intéresser à ce qui se passait sur les listes de diffusion KDE. Je considérais ces canaux avant tout comme des moyens de communiquer avec un public choisi, principalement des utilisateurs et des contributeurs du projet en tant que tel.

Pendant un certain temps, je ne réalisai pas que les articles que je publiais sur *The Dot* pourraient être repris par des journalistes. Je m'appliquais à les écrire parce que je voulais faire du bon boulot et non pas parce que j'avais peur de passer pour folle auprès du reste du monde. La liste de presse était maintenue par d'autres personnes et ce que j'écrivais ne me paraissait pas important non plus. Je voulais toucher certaines personnes. Pour cela les canaux officiels et mon propre blog me semblaient être les moyens les plus efficaces.

Être citée sur ReadWriteWeb[1] après avoir annoncé sur mon blog que je commencerai un nouveau boulot fut un choc pour moi. Non pas parce que j'ignorais que des gens lisaient ce que j'écrivais (j'espérais bien qu'ils le lisent !) mais je ne m'attendais pas à ce que ça soit un sujet d'une telle importance. Ce n'était même pas pendant les vacances d'été. Encore heureux que personne ne me l'ait dit, je n'aurais pas été capable de publier ne serait-ce qu'une seule ligne.

L'Étranger

Il y a quelque temps, quand j'ai assisté à ma première conférence, j'avais la ferme conviction que j'étais différente des autres

1 http://fr.wikipedia.org/wiki/ReadWriteWeb.

participants. Je me voyais comme une étrangère parce que je n'avais pas grand-chose en commun avec qui que ce soit à part un vague intérêt pour la technologie : je travaillais en *freelance* depuis quelques années déjà, après mon diplôme universitaire ; je n'avais aucune éducation pertinente dans le domaine, et j'étais mère d'un enfant de 10 ans. Sur le papier en tout cas, il ne pouvait pas y avoir plus éloignée des suspects habituels qu'on rencontre dans les projets FOSS.

En 2008, j'ai assisté à un sprint[1] KOffice au sein de l'équipe promotion et marketing de KDE pour préparer la sortie de la version 2.0. L'idée initiale était d'esquisser une série d'activités promotionnelles autour de cette sortie afin de développer à la fois les supports développeur et utilisateur. Pour ce dernier, nous étions trois à suivre un chemin parallèle à celui qui concernait les développeurs.

Nous avons essayé de comprendre comment nous pourrions positionner KOffice et adapter la communication au public ciblé. Très tôt dans le processus, nous avons découvert que nous devions faire marche arrière : à ce stade, le manque de maturité de la suite rendait impossible son positionnement comme un choix possible pour les utilisateurs non avertis. Nous devions nous en tenir aux développeurs et aux précurseurs. C'était difficile à vendre à certains développeurs, mais en tant qu'extérieurs au projet nous avions la chance d'appréhender le logiciel sans penser à tout le sang, la sueur et les larmes versés dans le code.

Pour beaucoup de projets, de n'importe quelle sorte, jeter un œil objectif à la situation donne du fil à retordre aux contributeurs principaux. Nous avons tendance à ne pas voir les grands succès quand nous sommes très concentrés sur des problèmes de détails et réciproquement. Parfois, nous manquons une occasion parce que nous pensons que ça n'a rien à voir avec ce que nous faisons

1 [NdT] phase de développement, généralement perçue comme intense, aboutissant à un produit fonctionnel.

(ou, pour commencer, parce que personne ne voudrait que ça ait quelque chose à voir).

Dans tous ces cas, les contributeurs extérieurs au projet disposent du potentiel pour apporter des points de vue différents à la discussion, particulièrement quand il s'agit de déterminer un ordre de priorité. C'est encore plus utile quand ce ne sont pas des développeurs : ils poseront différentes questions, sans ressentir de pression face à la connaissance et à la compréhension de tous les détails techniques ; ils peuvent aussi aider pour les décisions ou la communication sur un plan moins technique.

Conclusion

L'ignorance est une bénédiction. Ce n'est pas seulement vrai pour les individus qui profitent de l'insouciance qui en résulte mais aussi pour les projets que ces individus rejoignent. Ils apportent différents points de vue et expériences.

Et maintenant, filez et trouvez vous-même un projet qui vous intéresse, indépendamment de ce que vous pensez savoir.

Empaquetage

28. Vers l’activité professionnelle

Jonathan Riddell

Jonathan Riddell est développeur KDE[1] *et* Kubuntu[2]*, actuellement employé par* Canonical[3]*. Quand il n’est pas devant un ordinateur, il fait du canoë sur les rivières d’Écosse.*

Il y avait un bogue dans le code. Un bien méchant en plus : un plantage sans enregistrement des données. C’est bien là le problème. Dès qu’on regarde le code, on trouve des trucs à réparer. C’est facile de s’impliquer dans le logiciel libre ; le plus dur est d’en sortir. Après le premier bogue réparé, il y en a d’autres, et de plus en plus, tous à portée de main. Les corrections de bogues mènent à l’ajout de fonctionnalités, ce qui mène à la maintenance de projet, ce qui mène à faire fonctionner une communauté.

Ça a commencé en lisant Slashdot[4], cette masse d’actualité geek et technique peu filtrée, avec des commentaires de quiconque peut recharger la page assez vite pour être en haut de liste. Chaque actualité était intéressante et excitante, apportait un éclairage nouveau sur le monde de la technologie qui finissait par me fasciner. Je n’avais plus à accepter ce qui m’était donné par de

1 http://fr.kde.org.
2 http://www.kubuntu.org.
3 http://www.canonical.com.
4 http://slashdot.org.

grandes entreprises de logiciels, je pouvais voir là, dans la communauté du logiciel libre, le code se développer devant moi.

Quand j'étais étudiant, il était possible de finir très rapidement les exercices donnés par les professeurs. Mais les exercices ne sont pas des programmes terminés. Je voulais savoir comment appliquer les compétences basiques qu'on m'avait données dans le monde réel en écrivant des programmes résolvant des problèmes réels pour les gens. J'ai donc recherché du code, qui n'était pas difficile à trouver, il se trouvait là, sur Internet, en fait. En regardant le code des programmes que j'utilisais de plus près, j'y ai décelé de la beauté. Non pas parce que le code était parfaitement soigné ou bien structuré, mais parce que je pouvais le comprendre avec les concepts que j'avais déjà appris. Ces classes, méthodes et variables étaient bien en place, me permettant de résoudre les problèmes pertinents. Le logiciel libre est le meilleur moyen de franchir le pas entre savoir comment finir ses exercices de cours et comprendre comment de vrais programmes sont écrits.

Tous les étudiants en informatique devraient travailler sur du logiciel libre comme sujet de leur mémoire. Sinon, vous avez de grandes chances d'y passer six mois à un an pour que le mémoire en question finisse au sous-sol d'une bibliothèque sans être jamais plus consulté. Seul le logiciel libre permet d'exceller en faisant ce qui va de soi : vouloir apprendre comment résoudre des problèmes intéressants. À la fin de mon projet, des programmeurs de la NASA utilisaient mon outil de création de diagrammes en UML[1] et il reçut des prix au cours de réceptions somptueuses. Avec le logiciel libre, on peut résoudre de vrais problèmes pour de vrais utilisateurs.

La communauté des développeurs est remplie de personnes formidables, passionnées et dévouées à leur travail, sans autre espoir de récompense qu'un programme d'ordinateur couronné de succès. La communauté des utilisateurs est également incroyable.

1 [NdT] langage de modélisation unifié.

Il est satisfaisant de savoir qu'on a aidé quelqu'un à résoudre un problème. Et j'apprécie les messages de remerciement que je reçois.

Après avoir écrit un logiciel utile, il faut le mettre à la disposition du plus grand nombre. Le code source ne va pas fonctionner pour la plupart des gens, il doit être compilé. Avant d'être impliqué, je trouvais que le fait de compiler était une manière un peu paresseuse de contribuer au logiciel libre. Vous vous attirez la plus grande partie de la reconnaissance sans rien avoir à coder. C'est, quelque part, quelque chose d'injuste. De même, la gestion de la communauté nécessaire pour porter un projet de logiciel libre peut aussi être vue comme une façon de s'attirer la reconnaissance sans faire de code.

Les utilisateurs dépendent beaucoup des *packagers*[1]. Il est nécessaire que leur travail soit à la fois rapide, pour satisfaire ceux qui veulent la dernière version, et fiable, pour ceux qui veulent la stabilité (autant dire tout le monde). La partie la plus délicate, c'est que cela implique de travailler avec les logiciels des autres, qui sont toujours « cassés ». Une fois que le logiciel est lâché dans la nature, commencent à émerger des problèmes qui n'étaient pas repérables sur l'ordinateur de l'auteur. Il est possible que le code ne puisse pas être compilé avec une version de compilateur différente, peut-être que la licence n'est pas claire et ne permet pas de le copier, peut-être que la gestion des versions est incohérente et qu'une mise à jour mineure est incompatible, ou encore que la taille de l'écran est différente, les environnements de bureau peuvent aussi l'affecter, quelquefois des bibliothèques tierces nécessaires ne sont pas encore à jour. De nos jours, le logiciel doit pouvoir tourner sur différentes architectures. Les processeurs 64 bits ont occasionné pas mal de problèmes quand ils sont devenus courants. Aujourd'hui, ce sont les processeurs ARM qui déjouent les calculs des codeurs. Les *packagers*

1 [NdT] les « empaqueteurs » qui préparent et maintiennent les paquets logiciels.

doivent régler tous ces problèmes pour donner aux utilisateurs quelque chose qui fonctionne de façon fiable.

Nous avons une règle chez Ubuntu selon laquelle les paquets avec des tests unitaires doivent inclure ces mêmes tests dans le processus de la création des paquets. Souvent, ils échouent et l'auteur du logiciel nous dit que les tests sont uniquement à son usage. Malheureusement, quand il s'agit de logiciel, il n'est jamais assez fiable de le tester soi-même, il doit aussi être testé par d'autres. Un test unique est rarement suffisant, il faut une approche à plusieurs niveaux. Les tests unitaires du programme original devraient être le point de départ, ensuite, le *packager* le teste sur son propre ordinateur, il faut ensuite que d'autres personnes testent aussi. L'installation automatique et les tests de mise à jour peuvent être scriptés assez correctement sur les services d'informatique dans le nuage. L'envoyer dans la branche de développement d'une distribution permet d'effectuer plus de tests avant de distribuer le logiciel en masse quelques mois après. À chaque étape, des problèmes peuvent être et seront découverts, ils devront être corrigés, puis ces correctifs eux-mêmes devront être testés. Il n'y a donc pas forcément à écrire beaucoup de code, mais il y a pas mal de travail pour passer le logiciel de 95 % à 100 % prêt. Ces 5 % sont la partie la plus difficile, un lent et délicat processus qui demande une grande attention pendant tout son cours.

Vous ne pouvez pas faire de paquets sans une bonne communication avec les développeurs en amont. Quand des bogues se produisent, il est vital de pouvoir trouver les bonnes personnes auxquelles parler rapidement. Il est important d'apprendre à bien les connaître comme on connaît des amis et des collègues. Les conférences sont vitales pour cela, car rencontrer quelqu'un apporte beaucoup plus de contexte à un message sur une liste de diffusion qu'une année entière de messages.

Une des faces cachées du monde du logiciel libre réside dans la communication par les canaux IRC[1] privés utilisés par les principaux membres d'un projet. Tous les grands projets en ont. Quelque part, Linus Torvalds a un moyen de discuter avec Andrew Morton et les autres sur ce qui est bon et sur ce qui est mauvais dans Linux. Ils sont plus sociaux que techniques et, quand on en abuse, ils peuvent être très antisociaux pour la communauté en général. Mais pour les moments où on a besoin d'un canal de communication rapide sans bruit parasite, ils fonctionnent bien.

Tenir un blog est un autre moyen de communication important dans la communauté du logiciel libre. C'est notre principale méthode pour promouvoir à la fois le logiciel que nous produisons et nous-mêmes. Non pas que ce soit utilisé éhontément pour de l'auto-promotion (il est inutile de prétendre que vous sauverez des vies avec votre blog…), mais parler de votre travail sur le logiciel libre aide à construire une communauté. Cela peut même vous valoir de trouver un travail ou d'être reconnu dans la rue.

Ces histoires venant de Slashdot, à propos de développements de nouvelles technologies, ne concernent pas des personnalités éloignées que vous ne rencontrerez jamais comme dans la presse *people*. Elles concernent des personnes qui ont trouvé un problème et qui l'ont résolu en utilisant l'ordinateur qu'elles avaient en face d'elles. Pendant quelques années, j'ai édité le site d'informations de KDE, trouvant les personnes qui résolvaient des problèmes, concevaient des idées novatrices, s'acharnaient longuement à améliorer un logiciel jusqu'à ce qu'il atteigne une qualité suffisante, et j'en parlais au monde entier. Je n'ai jamais été à court d'histoires à raconter ni de personnes à présenter à tout le monde.

Mon dernier conseil est de conserver de la diversité. Il existe une incroyable richesse de projets intéressants à explorer, qui vous permettent d'apprendre et de progresser. Mais une fois que

1 http://fr.wikipedia.org/wiki/Internet_Relay_Chat.

vous avez atteint une position de responsabilité, il peut être tentant d’y rester. Après avoir aidé à créer une communauté pour Kubuntu, je repars temporairement vers un travail sur Bazaar, un projet très différent, orienté sur les développeurs plutôt que sur des utilisateurs novices en technologies. Je peux à nouveau apprendre comment le code devient une réalité utile, comment une communauté communique, comment la qualité est maintenue. Ce sera un défi amusant et j’ai hâte de m’y attaquer.

29. Empaqueter : la voie royale du logiciel libre

Thorn May

Thom May est développeur Debian et membre émérite de la fondation Apache. Il a été parmi les premiers à être engagé chez Canonical, l'entreprise mère d'Ubuntu. Il vit aujourd'hui à Londres et dirige l'unité de développement chez MacMillan Digital Science.

Introduction

Voilà plus de dix ans que je me suis lancé dans le logiciel libre. J'avais utilisé Debian pendant quelques années à l'université et décidé que je voulais donner quelque chose en retour. J'ai donc commencé un long voyage pour franchir les étapes permettant de devenir un « nouveau responsable Debian »[1] sans avoir jamais vraiment contribué au logiciel libre auparavant et inquiet qu'un manque d'expérience en C pourrait se révéler être un gros problème.

Il s'avéra que cette inquiétude était largement infondée. En commençant à travailler avec des paquets que j'utilisais régulièrement, j'ai pu contribuer efficacement. En même temps que mon

1 http://www.debian.org/doc/manuals/maint-guide/index.fr.html.

expérience avec la myriade d'outils et de systèmes que Debian fournit à ses mainteneurs s'accroissait, je devenais plus efficace pour gérer mon temps et j'étais capable de travailler sur un ensemble de paquets plus étendu.

M'occuper de davantage de paquets m'amena à travailler avec un ensemble plus important de systèmes de compilation, de langages de programmation et de boîtes à outils. Cela m'aida également à m'intégrer à la communauté Debian. Aussi rugueuse et dogmatique soit-elle, la communauté Debian repose sur des mainteneurs doués et expérimentés : c'est l'une des raisons principales pour lesquelles Debian a maintenu son excellence technique sur une si longue période.

À peu près à cette époque, le projet Apache httpd approchait enfin des premières versions bêta de httpd 2.0 qui était restée des années en chantier et était sur le point de connaître une mise à jour majeure. L'équipe Apache de Debian avait été plutôt inactive depuis quelque temps – les paquets de la version 1.3 étaient stables et changeaient peu – et n'avait pas prévu d'empaqueter la version 2.0. J'ai pris un vif intérêt au travail nécessaire pour garantir la bonne maintenance des paquets httpd. Je travaillais comme administrateur système en charge de nombreux serveurs web Apache, il tombait donc sous le sens que je devais relever le défi de la production de paquets pour la nouvelle version.

Avec un ami, nous avons commencé le travail sur les paquets et nous avons rapidement découvert que, alors que le code approchait le niveau de qualité d'une bêta toute fraîche, l'outillage autour de la compilation et de la personnalisation de httpd était hélas manquant, ce qui est assez représentatif de bien des projets logiciels complexes.

Pendant pratiquement toute une année – alors que les développeurs en amont stabilisaient leur code et qu'un nombre croissant d'utilisateurs précoces commençaient à tester et à déployer la nouvelle version – nous avons travaillé dur pour garantir que le

système de compilation soit suffisamment flexible et robuste pour satisfaire aux rigoureux prérequis de la politique de Debian. Nous devions garantir que nos paquets étaient techniquement corrects, nous assurer que notre relation avec les développeurs en amont nous permettait de leur faire remonter des correctifs dès que possible, que nous serions prévenus des tests précoces des versions candidates et alertés dès que des problèmes de sécurité apparaîtraient.

Mes interactions avec Apache pendant l'empaquetage et la maintenance de httpd 2.0 m'ont amené à m'engager en amont du projet, ce qui signifiait que je pouvais directement contribuer au code. C'est, en général, la dernière étape avant de passer de l'empaquetage d'un logiciel à son développement actif à destination d'un public plus large que celui de votre distribution. À titre personnel, cette reconnaissance m'a donné la confiance pour contribuer à bien plus de projets libres puisque je savais que mon code était de qualité suffisante pour être bien accueilli.

L'évolution – du *packager* au développeur

Comment est-ce arrivé ? La création de paquets, dans sa forme la plus simple, permet d'assurer qu'un projet logiciel donné se conforme à la politique de la distribution ; dans mon cas, Debian. De manière générale, cela signifie configurer le logiciel au moment de la compilation afin qu'il soit installé dans les répertoires idoines spécifiés par le *Filesystem Hierarchy Standard*, ou FHS[1], que les dépendances aux autres paquets soient correctement spécifiées et que le logiciel fonctionne normalement sur la distribution.

La création de paquets complexes peut nécessiter la division d'un projet en amont en de multiples paquets. Par exemple, les bibliothèques et les fichiers d'en-tête permettant à l'utilisateur de compiler le logiciel avec leur bibliothèque sont fournis dans des paquets distincts, et des fichiers dépendant de la plate-forme

1 [NdT] Norme de la hiérarchie des systèmes de fichiers.

peuvent être fournis séparément de ceux qui en sont indépendants. S'assurer que le logiciel en amont se déploie correctement dans ces situations nécessitera souvent des changements dans le code. Ces changements sont la première étape vers un travail actif sur un projet, plutôt que le travail parfois passif de création de paquet.

Une fois que votre paquet est disponible dans la distribution, il est exposé à des millions d'utilisateurs potentiels. Ces utilisateurs vont sans aucun doute exécuter votre logiciel selon des pratiques que ni vous, en tant que *packager*, ni vos développeurs en amont n'aviez prévues. Sans surprise, avoir de nombreux utilisateurs implique l'arrivée de nombreux rapports de bogue. Debian, comme la plupart des distributions, encourage ses utilisateurs à lui soumettre directement les rapports de bogue plutôt qu'à chacun des projets individuels en amont. Ceci permet aux mainteneurs de trier les rapports de bogue et d'assurer que les modifications effectuées lors du processus de création de paquet ne sont pas la cause du problème rapporté. Souvent, il peut y avoir un grand nombre d'interactions entre le rapporteur du problème et le mainteneur du paquet avant que les développeurs en amont ne soient impliqués.

Au fur et à mesure que le mainteneur du paquet accroît sa connaissance du projet, il sera en mesure de résoudre directement la plupart des problèmes. Le mainteneur publiera souvent des correctifs de bogue directement dans Debian tout en les faisant remonter en amont, permettant ainsi à la fois une résolution rapide des problèmes et de nombreux tests de correctifs. Une fois qu'un correctif est validé, le mainteneur travaillera alors avec le projet amont pour s'assurer que les changements requis y ont bien lieu, de manière à ce qu'ils soient disponibles aux autres utilisateurs du logiciel.

Fournir des correctifs de bogue réussis pour des distributions telles que Debian relève souvent d'une forme complexe d'art. Debian fonctionne sur de nombreuses plates-formes, allant des

gros serveurs IBM aux smartphones, et la gamme ainsi que l'étendue de ces plates-formes révèlent rapidement les approximations dans le code. L'empaqueteur a, la plupart du temps, un accès plus aisé à cette série de plates-formes que les développeurs en amont et constitue, de ce fait, le premier point d'appel quand un problème épineux de portage apparaît. On apprend rapidement à reconnaître les symptômes d'une approximation de la taille d'un pointeur, les problèmes avec les endianness[1] et bien d'autres problèmes ésotériques ; cette expérience permet de devenir un programmeur à la fois plus polyvalent et plus prudent.

Lorsqu'un paquet reçoit des corrections de bogues et des améliorations, il est essentiel que ces changements remontent en amont. Trop souvent, la différence entre un paquet et le logiciel définitif en amont peut s'accroître énormément, avec pour conséquence que les deux bases de code deviennent presque entièrement distinctes. Non seulement cela alourdit la tâche de la maintenance des deux côtés, mais cela peut aussi créer une immense frustration et faire perdre beaucoup de temps en amont dans le cas où un utilisateur de votre paquet rapporte un bogue lié à l'un des changements dans la version empaquetée en amont. Il est par conséquent vital que s'établissent une relation de travail étroite avec la branche amont et un consensus sur la meilleure façon de collaborer entre les deux parties.

La collaboration entre les développeurs et le *packager* peut prendre bien des formes. Que ce soit trouver la bonne voie pour communiquer les rapports de bogue, s'assurer de l'utilisation du bon style de codage, du même usage du système de contrôle de version ou des interactions qui provoquent le moins de frictions possible. Tout ceci amène une bien meilleure relation avec l'amont et accroît grandement la probabilité que ceux qui y travaillent prendront le temps de vous aider quand vous en aurez besoin.

1 https://fr.wikipedia.org/wiki/Endianness.

Une fois que la relation de travail entre l'amont et vous est établie, il devient facile de contribuer plus directement en amont. Ceci peut également se faire de bien des façons différentes. Les premières étapes, simples, peuvent impliquer la synchronisation de n'importe quel rapport de bogue en amont avec ceux de votre distribution, afin d'être sûr que ce double effort impacte la cause primaire et résolve des bogues. Une implication plus directe consiste à développer des fonctionnalités et à changer plus largement que ce qui serait acceptable dans le cadre d'une version empaquetée.

Conclusion

Je pense que les deux choses essentielles que j'aurais aimé connaître lorsque j'ai commencé sont le sens de la communauté que le logiciel libre fait naître et la merveilleuse voie que le *packaging* de logiciel libre ouvre vers le plus vaste monde du logiciel libre.

La communauté est essentielle au succès du logiciel libre. Elle se présente sous différentes formes, de la multitude d'utilisateurs souhaitant investir du temps dans l'amélioration de votre logiciel, jusqu'aux pairs d'une distribution ou d'un projet logiciel, qui investissent leur temps et leur énergie à affûter vos compétences et à s'assurer que vos contributions sont aussi bonnes que possible.

La voie qui part du *packaging* pour aller vers le développement est l'une des plus empruntées. Elle présente une courbe d'apprentissage moins raide qu'aborder directement le développement et permet d'acquérir des compétences à un rythme moins soutenu qu'en suivant d'autres chemins.

30. Au confluent de l’amont et de l’aval

Vincent Untz

*Vincent Untz est un activiste passionné du logiciel libre, un partisan convaincu de GNOME, ainsi qu’un élément moteur d’*openSUSE[1]*. Il a été responsable des versions de GNOME de 2008 à 2011, jusqu’à la sortie de GNOME 3.0 ; il a été directeur exécutif de la fondation GNOME (2006-2010) et il dirige l’équipe GNOME chez openSUSE. Quoi qu’il en soit, il trouve plus simple de se décrire comme un « touche-à-tout » et il travaille dans divers services (certains diraient au petit bonheur la chance) du bureau pour aider openSUSE à rester extraordinaire. Vincent continue à faire du forcing pour que le français soit la langue officielle de GNOME et compte bien y parvenir bientôt. Sinon, il aime la crème glacée.*

Il y a bien longtemps, dans une chambre, la nuit…

J’étais en train de jeter un dernier coup d’œil sur une liste de bogues pour voir si je n’avais pas oublié de fusionner un correctif. Je m’étais bien assuré d’écrire ce que je pensais être une entrée NOUVEAUTÉS au sujet de la nouvelle version. J’ai entré `make distcheck`[2] et je regardais la console afficher des centaines de

1 http://fr.opensuse.org/Bienvenue_sur_openSUSE.org.

2 [NdT] commande GNU permettant de créer un paquet et de le tester automatiquement dans un répertoire différent de celui de développement pour démarrer le processus de diffusion.

lignes. Une archive avait été créée, et j'ai à nouveau vérifié que l'archive se construisait correctement. J'ai vérifié, encore et encore : j'étais inquiet. D'une certaine manière, je ne faisais pas totalement confiance à la commande `make distcheck`. Après avoir tout vérifié plusieurs fois, j'ai envoyé l'archive sur le serveur et expédié un courriel d'annonce.

J'avais réussi à le faire : j'avais sorti ma première archive d'un logiciel sur le développement dont j'étais récemment devenu co-responsable. Et j'ai certainement pensé : « Ah, maintenant les utilisateurs vont pouvoir apprécier un bon truc ! ». Mais à peine quelques secondes après le chargement de mon archive, quelques personnes l'ont téléchargée et ont rendu ma version réellement accessible aux utilisateurs.

C'est une chose que je tenais pour acquise, car je pensais que c'était une tâche banale. J'avais tort.

Amont et aval

En tant qu'utilisateurs, nous ne comprenons pas forcément les différentes étapes nécessaires pour acheminer un logiciel jusqu'à nous. Il est là, et nous pouvons nous contenter d'en profiter.

De nombreuses personnes participent au processus d'acheminement du logiciel. Et cet effort se répartit généralement entre deux groupes de personnes d'égale importance dans la manière dont fonctionne le logiciel libre aujourd'hui :

– En amont : c'est le groupe qui crée le logiciel. Il inclut évidemment les programmeurs mais, en fonction du projet, d'autres catégories de contributeurs sont également essentielles : designers, traducteurs, rédacteurs de documentation, testeurs, trieurs de bogues, etc. En général, l'amont se charge seulement de livrer le code source sous forme d'archive.
– En aval : c'est le groupe responsable de la distribution du logiciel aux utilisateurs. Tout comme en amont, les contri-

> buteurs ont une gamme de profils très variée et travaillent à la traduction, à la documentation, aux tests, au triage des bogues, etc. Il y a cependant un profil qui, jusqu'à présent, est spécifique au travail en aval : le *packager*, qui prépare le logiciel afin de le rendre disponible sous forme de paquet, un format plus adapté à un usage facile que le seul code source.

Chose intéressante, les utilisateurs ont aussi une bonne intuition de cette séparation, bien que nous n'en soyons pas conscients : nous supposons souvent que les développeurs de logiciels sont inaccessibles et nous envoyons plutôt nos retours et demandes d'aide aux distributeurs.

Pour éclairer cette séparation entre amont et aval, une analogie parlante et classique est celle du circuit des biens de consommation, avec les magasins de détail (« l'aval ») qui distribuent des produits manufacturés (« l'amont ») et jouent un rôle important pour les clients (« les utilisateurs »).

Un regard plus attentif sur l'aval

Si je devais résumer le rôle de l'aval en une phrase, voici comment je le décrirais :

> L'aval est *le pont* entre les utilisateurs et l'amont.

Lorsque j'ai sorti ma première archive en amont, je supposais que, pour l'aval, le travail consisterait principalement à compiler la source pour construire un paquet avec, et rien d'autre. Construire un paquet est effectivement la première étape. Mais c'est seulement le début du voyage vers l'aval : différentes tâches viennent ensuite. Certaines sont purement techniques tandis que d'autres sont sociales. Je vais me contenter de décrire très briève-

ment ce voyage ici, de manière non exhaustive, puisque ça pourrait faire l'objet d'un chapitre entier de ce livre[1].

La construction du paquet proprement dit peut se révéler moins triviale que prévu. Il n'est pas rare qu'un *packager* rencontre des problèmes qui étaient inconnus de l'amont. Comme lorsqu'une nouvelle version du compilateur est utilisée (avec de nouvelles erreurs), qu'une bibliothèque spécifique a d'abord besoin d'être mise à jour (parce que l'archive utilise de nouvelles interfaces de programmation) ou bien que le système de compilation de l'archive est conçu pour une certaine façon de fonctionner (qui ne suit pas les directives de la distribution cible). Ce qui est encore plus méconnu par beaucoup est le fait que tous ces problèmes peuvent également se produire après que l'archive a déjà été *empaquetée*, comme lors de la migration d'une distribution entière vers un nouveau compilateur ou bien une nouvelle chaîne de compilation. Aucun de ces problèmes techniques n'est vraiment compliqué à résoudre en lui-même, et l'amont est souvent content de contribuer à la solution. Mais sans l'aval, ces problèmes pourraient ne pas être remarqués par l'amont avant un long moment.

Ce qui selon moi est plus important que ces défis techniques, c'est que l'aval est généralement en contact avec davantage d'utilisateurs que l'amont. Cela se traduit par des rapports de bogue, des demandes de support, des requêtes de changement de la configuration par défaut ou bien d'autres choses encore. C'est là que la foule en aval donne la mesure éclatante de sa force : au lieu de simplement transférer ça en amont, l'aval va travailler sur les retours des utilisateurs afin de ne renvoyer que des synthèses qui seront utilisables en amont. Bien souvent, les rapports de bogue sont soumis avec trop peu d'informations sur le problème (auquel cas l'aval demandera plus de détails). Souvent, les demandes de support sont issues d'une incompréhension du côté

1 [Note de l'auteur] je tiens à préciser que je ne crois pas que l'aval devrait modifier significativement le logiciel mis à disposition par l'amont. Certains, en aval, le font tout de même et cela s'ajoute à leur charge de travail.

de l’utilisateur (ce que l’aval peut, parfois, traduire par une suggestion visant à modifier le programme afin d’éviter cette incompréhension). Souvent encore, de nouveaux paramètres par défaut sont suggérés sans réflexion suffisante (l’aval travaillant alors avec les utilisateurs pour voir si le raisonnement est valide). À partir de cette énorme quantité de données, l’aval produira un ensemble d’informations plus compact, que l’amont sera en mesure de digérer facilement. Ce qui amènera à des améliorations sur le logiciel.

Il existe généralement deux récompenses pour les contributeurs en aval : les contributions directes et indirectes vers le projet en amont grâce aux efforts effectués par l’aval sont suffisantes pour beaucoup. Mais plus important encore, le contact direct avec davantage d’utilisateurs amène à recueillir la satisfaction qu’ils expriment. Et une situation aussi gratifiante rend facilement heureux beaucoup de gens.

Une petite note au passage : lorsqu’on considère la quantité de travail fournie en aval, je ne serais pas surpris qu’au bout du compte, beaucoup de contributeurs en amont soient bien contents d’avoir des gens agissant comme intermédiaires : cela diminue significativement la quantité de retours tout en améliorant leur qualité (et en évitant les commentaires en double, les problèmes non documentés, etc.). Cela permet à ceux qui travaillent en amont de rester focalisés sur le développement lui-même, au lieu de les obliger à soit trier les retours, soit les ignorer.

Rien qu’en regardant mon expérience en amont, je ne compte plus le nombre de correctifs que j’ai reçus de l’aval pour résoudre des problèmes de compilation. Je me rappelle aussi d’innombrables discussions que j’ai eues à propos des bogues qui affectaient le plus les utilisateurs et qui m’ont permis de prioriser mon travail. De fait, depuis que j’ai rejoint les équipes en aval, j’ai commencé à faire remonter des correctifs proches de ceux liés à des problèmes de compilation à l’amont et à discuter avec ma base en aval pour transmettre des retours d’expérience d’utilisa-

teurs. Une telle collaboration amont-aval participe à l'amélioration de la qualité générale de notre écosystème du logiciel libre et je la considère comme essentielle à notre bonne santé.

Remonter de l'aval vers l'amont !

Je crois fermement que, pour qu'un projet réussisse, il faut qu'il y ait une forte collaboration entre amont et aval. Je doute que beaucoup désapprouvent. Cependant, par « aval », les gens pensent généralement au travail fait dans les distributions. Mais, particulièrement pour les applications, il devient de plus en plus pertinent de pousser ce travail fait en aval en dehors des distributions et de tirer parti d'un tel mouvement vers l'amont.

Des outils comme l'Open Build Service[1] permettent plus facilement d'avoir des personnes qui compilent et distribuent des paquets d'une application pour plusieurs distributions. Cela présente des avantages à la fois pour les utilisateurs (qui peuvent profiter plus rapidement et plus facilement des mises à jour de leurs applications préférées) et pour l'amont (qui peut aider à construire une relation plus forte avec sa base d'utilisateurs). Le seul défi qu'un tel mouvement représente est le besoin perpétuel d'avoir quelqu'un qui s'occupe de l'empaquetage, mais aussi qui gère des retours plus nombreux des utilisateurs. Dans les faits, il y a toujours besoin de quelqu'un pour faire le travail de l'aval, sauf qu'il serait fait au sein de la branche amont.

Pour moi, cela semble une perspective excitante et j'irais même plus loin en suggérant que nous, la communauté du logiciel libre, devrions migrer lentement le travail d'aval fait sur les distributions vers un travail d'aval fait directement, aussi souvent que possible, en amont. C'est souvent possible, au moins pour les applications. Cela requiert évidemment de penser différemment. Mais ça permettrait de partager un travail qui, actuellement, est le plus souvent dupliqué sur toutes les différentes branches en aval.

1 http://openbuildservice.org.

Pour ceux qui souhaitent actuellement commencer à contribuer aux applications qu'ils apprécient, ce travail sur les paquets en amont est une toute nouvelle approche qui pourrait vraiment être une réussite !

J'ai essayé, je suis resté. Pourquoi pas vous ?

L'aval a toujours été essentiel à ma vie en tant qu'utilisateur de logiciels libres – après tout, seules quelques personnes installent manuellement leur système à partir de zéro et je n'en fais pas partie. Cependant, c'est aussi devenu un atout pour moi en tant que développeur en amont, étant donné que j'ai commencé à prendre plus de temps pour discuter avec des personnes en aval afin d'obtenir plus de retours sur les bogues, les fonctionnalités, la qualité générale et même les directions futures du logiciel sur lequel je travaillais.

C'est seulement lorsque j'ai commencé à être moi-même en aval que j'ai compris que cette position est en effet privilégiée afin de conseiller en amont, grâce au contact direct avec les utilisateurs et la perspective différente que l'on retient de cette position différente.

Sans l'aval, nous ne serions pas là où nous en sommes aujourd'hui. Si vous souhaitez avoir un impact significatif, soyez persuadé qu'en participant en aval et en discutant avec l'amont, vous réussirez.

Et vous y prendrez du plaisir.

Promotion

31. Trouver ses marques dans l’équipe

Stuart Jarvis

Stuart Jarvis a commencé à travailler avec l’équipe de promotion de KDE en 2008 en écrivant des articles pour le site web d’actualités de KDE[1]*. Il a appris à la dure comment faire bouger les choses dans une communauté du logiciel libre et participe davantage aux activités de l’équipe de promotion en écrivant les annonces des nouvelles versions de KDE et en rédigeant des articles sur les logiciels KDE dans la presse Linux. Il siège maintenant dans le groupe de travail marketing de KDE, contribue à définir la ligne de conduite pour la promotion de KDE et les activités marketing. Il aide aussi les nouveaux contributeurs à trouver leurs marques. Il fait maintenant aussi partie de l’équipe éditoriale de KDE.-News, là où il a commencé à participer.*

« Qui code décide » est le mantra du développement dans le logiciel libre. Mais que faire quand il n’y a pas de code ?

Rejoindre l’équipe de promotion et de marketing de votre projet de logiciel libre préféré représente un défi particulier. Pour les nouveaux codeurs, la plupart des projets ont des systèmes de révision du code, des mainteneurs et des pré-versions du logiciel qui les aident à mettre en évidence les erreurs dans le code, ce qui rend moins effrayante la contribution à leur premier correctif.

1 http://news.kde.org.

La promotion en revanche peut nécessiter que votre travail soit visible par le public, après une relecture minimale, parfois immédiatement. La nature non hiérarchisée des communautés de logiciel libre implique qu'il y ait rarement une unique personne vers laquelle vous pouvez vous tourner et qui pourra vous dire si vos idées sont bonnes et prendre des responsabilités à votre place.

Obtenir un consensus versus obtenir des résultats

J'ai d'abord commencé à contribuer à KDE en écrivant des articles pour le site d'actus officiel, KDE.News. J'avais déjà écrit pour des organes de presse, mais j'avais toujours affaire à une personne bien identifiée à qui j'envoyais un brouillon pour avoir un retour et faire les corrections demandées. Dans l'équipe de promotion de KDE, il n'y avait pas une seule personne ou un seul groupe de personnes pour assumer cette tâche. Je devais essayer, juger aux réponses que j'avais sur les brouillons d'articles, décider si j'avais tous les retours qui m'étaient nécessaires et si l'article était prêt pour une publication.

Avec les conseils de contributeurs plus expérimentés, j'ai finalement appris à proposer quelque chose et à le publier en quelques jours s'il n'y avait pas d'objection majeure. Cette approche peut être utilisée par n'importe quel contributeur d'une équipe de promotion de logiciel libre, qu'il soit débutant ou confirmé.

Tout d'abord, réfléchissez à la façon dont vous allez vous y prendre, que ce soit pour écrire un article, changer le texte d'un site web ou donner une conférence dans votre école locale. Planifiez, écrivez l'article ou le nouveau texte. Envoyez vos idées, pour relecture, sur la liste de diffusion de l'équipe de promotion de votre organisation. Surtout, ne demandez pas aux gens ce qu'ils en pensent – vous pourriez attendre des jours ou des semaines sans obtenir de réponse définitive. Signalez plutôt que vous allez publier ou soumettre votre texte, ou mettre en œuvre votre programme à telle date précise, en attendant les objections d'ici là.

Lorsque vous soumettez une date limite, pensez au temps nécessaire à chaque membre actif de l'équipe pour lire ses messages et évaluer votre proposition. Vingt-quatre heures est un minimum absolu pour un simple oui ou non en réponse à une question fermée. Lorsqu'il s'agit de quelque chose nécessitant une lecture ou une recherche, vous devriez envisager un délai de réponse de plusieurs jours.

Si vous ne rencontrez pas d'objection majeure dans le délai que vous avez fixé, vous pouvez avancer. S'il existe de gros problèmes par rapport à votre projet, quelqu'un vous le dira. Les choses se font, en réalité. Vous ne serez pas frustré par un manque de progrès et vous aurez la réputation de mener à bien les tâches.

Finalement, c'est vous qui décidez

Les communautés du logiciel libre peuvent facilement devenir des groupes de discussion. Tout le monde a son opinion. Si vous n'êtes pas prudent, les discussions peuvent s'éterniser, s'évanouir au fur et à mesure que les personnes s'en désintéressent et finir sans conclusion convaincante. Cela peut paraître assez difficile à gérer lorsque vous faites partie de la communauté depuis quelque temps : vous avez l'habitude de prendre vos propres décisions et d'avoir votre propre idée sur ceux dont les avis vous importent. Quand vous débutez, cela peut être très déroutant.

Si vous voulez que votre propre travail aboutisse, vous allez probablement devoir faire des choix entre des points de vue opposés. Vous pouvez mettre un terme au débat en donnant un résumé des points principaux et en exposant votre opinion sur ces points. Essayez de ne pas laisser de questions ouvertes en suspens, à moins que vous ne souhaitiez un débat plus long – donnez simplement vos conclusions et dites ce que vous allez faire. Dès lors que vous êtes correct, les autres personnes respecteront probablement votre avis, même si elles ne sont pas d'accord.

Soyez proactif – n’attendez pas qu’on vous demande

Le premier contact avec l’équipe de promotion que vous voulez rejoindre peut très bien être l’envoi d’un courriel sur sa liste de diffusion pour proposer vos compétences. Je pensais pouvoir énumérer mes points forts et espérer que les gens me suggéreraient des choses à faire. En pratique, ça ne fonctionne pas tout à fait comme ça.

La plupart des communautés manquent de volontaires et ont vraiment besoin de vos compétences. Mais comme elles manquent de volontaires, elles peuvent aussi manquer de temps pour donner de bons conseils et encadrer. Si vous voulez travailler sur une partie spécifique du projet, dites-le. Il est beaucoup plus facile pour quelqu’un du projet de répondre simplement « Vas-y ! » plutôt que d’essayer de vous proposer un projet qui corresponde à vos compétences.

Même quand vous avez travaillé sur quelques projets et prouvé vos compétences, il y a peu de chances que vous soyez contacté directement pour une tâche. Ceux qui coordonnent l’équipe marketing ne connaîtront pas votre situation personnelle et peuvent donc être mal à l’aise à l’idée de vous demander quelque chose de particulier sur votre temps libre, gratuitement. Une communauté idéale va poster régulièrement – que ce soit sur une liste de diffusion ou une page web – les tâches que des volontaires peuvent prendre en charge. Si ce n’est pas le cas, trouvez vous-même des choses à faire et prévenez la liste de diffusion que vous êtes en train de les faire. Les gens vont le remarquer et cela augmente les chances que vous soyez directement contacté à l’avenir.

Si vous êtes proactif, vous pouvez rapidement vous rendre compte que vous êtes l'une des personnes expérimentées de la communauté vers lesquelles les nouveaux venus se tourneront pour avoir des conseils ou du travail à réaliser. Essayez de vous souvenir comment c'était quand vous avez commencé et faites en sorte de faciliter au maximum leur vie de nouveau contributeur.

32. Progresser à petits pas

Jos Poortvliet

Jos Poortvliet travaille en tant que gestionnaire de communauté pour SUSE Linux. Auparavant, il était actif dans la communauté KDE internationale en tant que responsable de l'équipe marketing. Dans sa « vie hors-ligne », il a travaillé dans différentes entreprises en tant que conseiller en stratégie d'entreprise. Il passe son temps libre à expérimenter dans sa cuisine, où il tente de parvenir à quelque chose de comestible.

> « Mieux vaut faire beaucoup de petits pas dans la bonne direction qu'un grand bond en avant pour retomber en arrière. » (vieux proverbe chinois)

Une idée géniale…

Il était une fois, au sein de l'équipe marketing d'un projet de logiciel libre, quelqu'un qui eut une idée géniale pour que le projet prenne davantage d'ampleur. Un programme serait mis en place pour permettre à des étudiants en informatique de prendre connaissance du projet et de le rejoindre. Des universités seraient contactées et quelqu'un s'adresserait à elles pour susciter leur intérêt. Des ambassadeurs iraient alors dans ces universités pour y donner des cours et encadrer les premiers pas des étudiants dans le monde du logiciel libre. Une fois qu'ils auraient rejoint le projet en ligne, ils seraient encadrés sur des tâches simples et

deviendraient finalement des contributeurs chevronnés ! Les universités adoreraient bien sûr ce programme. Avec un peu de chance elles commenceraient à participer plus activement, en donnant à leurs étudiants du code à écrire pour le projet et bien plus encore.

… qui n’a pas fonctionné…

J’ai vu l’idée développée dans la fiction ci-dessus sous bien des formes dans de nombreuses communautés et projets. C’est une idée géniale et d’un fort potentiel ! Nous savons tous qu’il faut commencer tôt – nos concurrents du logiciel propriétaire sont très bons pour ça. Nous savons également que nous disposons de suffisamment d’arguments pour convaincre les universités et les étudiants de participer – le logiciel libre et *open source* représente le futur, il offre de très belles possibilités de développement des compétences. Les compétences en programmation ou en administration sous Linux sont davantage demandées que des développeurs Java ou .NET ou que des administrateurs système Windows. Et surtout : c’est plus amusant. Quoi qu’il en soit, si vous allez dans des universités, vous ne verrez pas beaucoup d’affiches vous invitant à rejoindre des projets de logiciels libres. La plupart des professeurs n’en ont jamais entendu parler. Que s’est-il passé ? Permettez-moi de continuer mon histoire.

… pas à cause d’un manque d’efforts…

L’équipe en a discuté longtemps. D’abord en mettant des idées en commun – de nombreuses idées concernant la concrétisation du concept ont fusé. Le responsable de l’équipe les a rassemblées et mises sur le wiki. Un calendrier a été établi avec des échéances et le responsable a réparti les tâches, pour certaines parties. Certains ont commencé à rédiger des supports de cours, d’autres à lister les références des universités. Ils ont régulièrement demandé des suggestions et des idées sur la liste de diffusion et ont reçu beaucoup de réponses proposant d’autres supports de

cours, que le responsable a ajoutés à la liste des choses à rédiger. Tout devait être fait pendant le temps libre des volontaires, mais on pouvait toujours compter sur le responsable pour rappeler les échéances aux volontaires.

Après quelques mois, une structure était visible et de nombreuses pages étaient créées sur le wiki.

Entre-temps, néanmoins, le nombre de personnes impliquées depuis la discussion initiale a diminué, passant de plus de trente à environ cinq qui faisaient encore mine de travailler. Le responsable a décidé de revoir la feuille de route avec des dates butoirs et, après quelques appels lancés sur la liste de diffusion, dix nouveaux volontaires s’engagèrent à réaliser diverses tâches. Le rythme s’est à nouveau un peu accéléré. Un certain nombre de choses qui avaient déjà été faites ont dû être mises à jour et il y avait d’autres ajustements à faire. Malheureusement, les choses ont continué à s’aggraver et le nombre de personnes impliquées a continué à diminuer. Des *sprints* mensuels furent mis en place, et ils ont en effet abouti à terminer davantage de choses. Mais il y avait simplement trop à faire. Au bout d’environ un an, les dernières personnes actives ont jeté l’éponge. Il ne reste de tout cela que quelques ressources dépassées sur une page wiki obsolète...

… mais parce qu’elle était trop ambitieuse

Alors pourquoi ça n’a pas marché ? L’équipe avait pourtant appliqué les meilleures techniques de gestion de projet qu’on puisse trouver sur le Web : *brainstorming*, puis mise en place d’un planning avec un échéancier, des objectifs précis ainsi que des responsabilités. Ils ont fait ce qu’il fallait faire sur un projet bénévole : solliciter les personnes, les impliquer, donner la possibilité à chacun d’exprimer son opinion. Ça aurait dû fonctionner !

Ce ne fut pas le cas pour une raison simple : c’était trop ambitieux. C’est une tendance. Des idées géniales reçoivent beaucoup

de commentaires, sont inscrites dans de grands plannings qui se terminent en pages wiki incomplètes amenant à une faible implémentation qui finit par s'évanouir dans le néant.

Les responsables doivent admettre que la manière de travailler d'une équipe dans le domaine du logiciel libre et *open source* n'est pas la même que dans un environnement structuré et dirigé comme peut l'être une entreprise. Les gens ont tendance à être présents lorsque quelque chose d'excitant se produit, comme lors de la sortie d'une version majeure, puis à disparaître jusqu'au prochain gros événement. La création d'une équipe communautaire ne devrait jamais supposer que les gens resteront pleinement impliqués jusqu'à la fin. Il faut prendre en compte le fait qu'ils seront présents pendant un certain temps, puis s'absenteront durant de plus longues périodes avant de revenir. Les arrivées et les départs font qu'il y a beaucoup d'agitation superflue et que le travail avance lentement. Oui, il est possible de diriger des gens, mais il n'est pas possible de les gérer. Dès que vous apprenez à laisser l'aspect gestion de côté, vous pouvez davantage vous concentrer sur les choses à faire dans les plus brefs délais.

Par conséquent, au lieu de planifier de grands projets ambitieux, trouvez quelque chose de plus modeste qui soit réalisable et utile en soi. Non pas une page wiki avec un planning, mais la première étape de ce que vous voulez accomplir. Et ensuite donnez l'impulsion en faisant les choses. Faites le premier brouillon d'un article. Créez la première version d'un dossier. Copiez-collez à partir de n'importe quoi d'existant ou améliorez quelque chose qui existe déjà. Ensuite, présentez le résultat à l'équipe, même s'il n'est qu'au stade du brouillon, et demandez si quelqu'un souhaite l'améliorer. Faites d'abord une petite chose et ça fonctionnera.

Ne planifiez pas, agissez…

Comment alors allez-vous réaliser quelque chose d'aussi énorme qu'un programme de recrutement des étudiants avec le

concours des universités ? Ne le faites pas ! Du moins, pas directement. Il faut en discuter avec toute l'équipe et le planifier – ça donnera certainement lieu à une discussion sympathique pouvant durer des semaines. Mais ça ne vous mènera pas loin. Gardez plutôt le plan pour vous-même. Sérieusement.

Je ne suis pas en train de dire qu'il ne faudrait pas en parler – vous le pouvez. Faites part de votre ambitieux projet à tous ceux qui sont intéressés. Et c'est tant mieux s'ils font des propositions. Mais n'en attendez pas trop, ne faites pas de plans au-delà de la première ou des deux premières étapes. Allez plutôt de l'avant, construisez sur ce qui existe déjà. Envoyez le brouillon d'un support de communication fraîchement créé ou amélioré à la liste de diffusion. Demandez à quelqu'un qui a déjà donné un cours sur votre projet de partager son support et améliorez-le un peu. Qui sait, les personnes dont le travail vous sert de base pourraient vous venir en aide ! Ceux avec qui vous avez discuté de votre projet et qui partagent votre vision pourraient également vous aider. De cette façon, vous terminerez souvent quelque chose – un prospectus, un site web amélioré ou une présentation utilisable. Et les gens peuvent, peu à peu, commencer à les utiliser. Des ambassadeurs peuvent se rendre dans leur université en utilisant une des choses que vous avez déjà créées. Pour cela, ils auront certainement besoin de créer des éléments manquants – qui pourront ensuite se retrouver sur le wiki. Et vous progressez.

… et vous aurez votre château en Espagne !

En marketing communautaire, la bonne stratégie ne réside pas dans le wiki. Elle ne dépend pas d'un programme ni d'un planning. Elle n'est pas non plus discutée chaque semaine avec l'équipe complète. Elle fait partie d'une vision qui s'est développée au cours du temps. Elle est portée par quelques personnes-clés qui indiquent le planning à court terme ainsi que les objectifs et elle est partagée par l'équipe. Mais elle n'a pas de date butoir ni de risque d'échouer. Elle est flexible et ne dépend de rien ni de

personne en particulier. Et ça restera toujours un château en Espagne…

Donc, si vous voulez piloter un effort marketing pour une communauté de logiciel libre, faites en sorte que la vision d'ensemble reste une vision d'ensemble. Ne planifiez pas trop, mais arrangez-vous pour que les choses aboutissent !

33. Savoir vendre son projet

Sally Khudairi

Active sur le Web depuis 1993, Sally Khudairi est la publicitaire en embuscade derrière certaines des organisations et des standards les plus importants de cette industrie. Ancienne adjointe de Sir Tim Berners-Lee et championne toutes catégories de l'innovation collaborative, elle a aidé au lancement de The Apache Software Foundation[1] *en 1999 et en fut la première femme et membre élue non technique. Sally est vice-présidente chargée du marketing et de la publicité pour The Apache Software Foundation et directrice générale de HALO Worldwide, une société de conseil en communication pour des marques de luxe.*

Tout le monde contribue au marketing. Du PDG à la star des commerciaux, en passant par le gars qui répartit le courrier dans la boîte, chacun est un représentant de votre entreprise. Les technologies et les stratégies ont changé au fil des années mais une bonne communication reste primordiale. Au bout du compte, tout le monde vend quelque chose, et c'est un équilibre intéressant à trouver dans la publicité : qui vous êtes, ce que vous faites et ce que vous vendez sont souvent étroitement imbriqués. Quand les gens me disent qu'ils ne savent pas qui je suis, je leur demande s'ils ont entendu parler du W3C, d'Apache ou des Creative Commons.

1 http://fr.wikipedia.org/wiki/Apache_Software_Foundation.

La réponse habituelle est « bien sûr ! », ce qui me confirme que je fais bien mon boulot. Si vous savez qui ils sont et ce qu'ils font, tout va bien. Après tout, c'est le produit qui compte, pas le publicitaire. Je n'ai jamais cherché à être là : me faire les dents dans la communication à la naissance du Web n'était pas facile, mais grâce au ciel j'ai pu observer les autres et esquiver un certain nombre de torpilles. Après une forte montée en puissance et quelques projets très en vue, quel conseil pourrais-je partager avec un chargé de relations publiques en herbe, avec un porte-parole chevronné rompu à la pratique des médias, ou un technologue qui ose enfourcher le cheval ombrageux de la promotion, malgré ses ruades ?

N'oubliez jamais de vous manifester

Quand vous vendez votre histoire à la presse, souvenez-vous que les médias, eux aussi, ont quelque chose à vendre. Bien sûr, au plus haut niveau, le rôle d'un journaliste est de raconter une histoire irrésistible et convaincante – qu'elle soit vraie ou non, que les faits soient exacts ou non —, qu'elle réponde ou non à une éthique, c'est une autre question. Qu'il s'agisse d'attirer le lectorat, de fidéliser les abonnés ou de promouvoir les espaces publicitaires, eux aussi sont en train de vendre quelque chose. Votre boulot, c'est de les aider à faire le leur. À dire vrai, il est possible que certaines personnes n'aient jamais entendu parler de vous, même si vous êtes dans le métier depuis déjà pas mal de temps. Même si ce n'est pas le cas, ils peuvent ne pas savoir exactement qui vous êtes. Soyez clair sur ce que vous avez à offrir. Quelle est l'accroche pour la presse – quelle est la nouvelle ? Assurez-vous qu'elle est vraiment nouvelle. Soyez direct et venez-en rapidement au fait. Vous devez être prêt à répondre aux questions suivantes : « et alors ? », « en quoi ça pourrait m'intéresser ? » et « qu'est-ce qu'il y a là-dedans pour moi ? ». Cela veut dire que vous devez vous poser des questions sur vous-même et sur votre produit. Les gens achètent des idées, pas des produits. Faire la promotion des avantages de ce que vous lancez

vous aidera à améliorer vos chances d’obtenir une couverture médiatique. Faites un pas de côté : qu’êtes-vous vraiment en train de vendre ?

Jamais le vendredi

Le pire des jours pour lancer un nouveau site web, diffuser un communiqué de presse ou informer les médias, c’est le vendredi. La probabilité qu’il se passe quelque chose et que personne ne soit disponible pour gérer les retombées est plus importante que vous ne pouvez l’imaginer. J’en ai eu une cuisante expérience dès le début de ma carrière. J’avais lancé la nouvelle page d’accueil du W3C un vendredi soir puis quitté le bureau et embarqué dans un avion pour Paris. Comme je venais du monde de la publication commerciale sur Internet, utiliser un tag propriétaire ne me posait aucun problème à partir du moment où il faisait le travail. Faire de même sur le site internet d’une organisation vouée à l’interopérabilité, en revanche, n’était pas une bonne idée. En quelques minutes, des douzaines de messages arrivèrent, demandant comment la <balise-aujourd’hui-dépréciée> était arrivée sur notre site. Et non, ça n’était pas `blink`.

N’imaginez jamais que cela n’a aucune importance

La crédibilité est essentielle. Même si vous êtes surchargé de travail, dévoué corps et âme ou partout à la fois, vous ne pouvez pas empêcher l’heure de tourner. Essayez de produire autant que vos capacités vous le permettent et demandez de l’aide si vous le pouvez. Certaines échéances doivent être négociées, et beaucoup d’éditeurs peuvent s’accommoder d’un retard dans le calendrier mais cela n’aura probablement pas (autant) d’importance une fois l’urgence passée si vous n’êtes pas capable de finir le travail. Tout comme pour l’art, le développement de standards et la relecture-correction, le processus peut se poursuivre et recommencer *ad nauseam*. Tandis que la créativité ne peut pas être gérée par le temps, des dates butoir strictes obligent à tracer une limite à un

moment donné. Mais vous devez vous soucier des détails. Arrêtez-vous. Révisez tout et testez tous les liens. Assurez-vous que cela correspond parfaitement à la stratégie de la campagne ou de la marque. Les cycles de répétition font partie des grands principes structurants de la communication et le travail continuera à s'accumuler. Organisez-le et protégez votre réputation.

Allez-y seul

Il est important de faire confiance à votre intuition, particulièrement lorsque vous sortez des sentiers battus. Aux premiers jours du Web supercool et ultramoderne, tout le monde semblait s'en remettre aux stratégies habituelles des marques/relations publiques/marketing qui consistaient à faire des sites vitrines. Puis tout le monde « suivait le *leader* » (le *leader* est « le premier à l'avoir fait », dans de nombreux cas). Les tendances sont une chose, les attentes et les besoins de l'industrie en sont une autre : « c'est comme ça que tout le monde fait » ne veut pas dire que c'est bien pour vous, votre projet ou votre communauté. Ma carrière dans la communication a commencé lorsque j'ai renvoyé le sous-traitant que nous avions choisi et tout ramené en interne.

Nous avons été parmi les premières organisations à mettre une adresse URL sur notre plaquette commerciale et les premiers aussi à utiliser une URL comme source d'un communiqué de presse, alors que les agences de presse nous disaient que cela n'était pas conforme et pas dans les règles établies. Faites confiance à vos connaissances. Allez à contre-courant et bousculez les règles de manière responsable. Sachez vous différencier. Il est permis d'être un dissident tant que vous pouvez défendre vos idées.

Offrez vraiment des perspectives

Bon nombre des technologies dans lesquelles je suis impliquée finissent en produits au bout de trois à cinq ans. Ceci signifie que, dans bien des cas, il est difficile d'établir une comparaison quel-

conque avec un produit similaire. Il est donc crucial que vous expliquiez clairement votre position en utilisant le moins de jargon possible. La plupart des journalistes et analystes non-développeurs avec lesquels je suis en contact ne suivent pas les activités d'une communauté particulière au quotidien et ne savent pourquoi telle fonctionnalité est meilleure qu'une autre, même si c'est une évidence pour vous.

Dire qu'on va « privilégier la forme plutôt que le fond » est plus pertinent aujourd'hui que jamais. Forme. Fond. Je marque toujours une séparation à ce sujet lorsque je fais de la formation aux médias : présentez trop le fond ou trop la forme et votre campagne risque d'échouer. La perception est fondamentale et la cause de bien des conflits. Tout sur la forme = « branché + hyperbole » = « Ah, ces marketeux ! ». Tout sur le fond = « des zéros et des uns » = « Ah, ces geeks ! ». Il vous faut comprendre et pouvoir expliquer clairement quel est le problème que résout votre produit. En sachant mieux présenter le problème, vous pourrez mieux en expliquer la solution. Les détails accessoires, les anecdotes et les succès, voilà ce qui donne à la presse un moyen d'attirer l'attention de son lectorat. Vous devez savoir répondre à la question « qu'y a-t-il pour moi là-dedans ? », parce que c'est ce qui incite les journalistes à fouiller un peu plus dans votre histoire, qui, en retour, permet aux lecteurs d'en savoir plus sur vous. La forme répond à la question « qu'y a-t-il pour moi là-dedans ? ». C'est donc l'hameçon. Le fond, c'est comment vous y êtes parvenu.

Avoir des porte-parole sur la brèche

Ayez toujours quelqu'un de disponible pour parler à la presse. Oui, ça peut être vous, mais sachez qu'il y aura un moment où, même si vous avez une histoire bien planifiée à raconter, vous pourriez ne pas être disponible. Avec qui d'autre travaillez-vous ? Qui vous connaît ? Qui vous soutient ? Définir ces personnes et distribuer les rôles pour clarifier qui dit quoi contribue beaucoup

à diminuer les maux de tête potentiels. J'agis habituellement en tant que porte-parole *d'arrière-plan* afin de pouvoir passer du temps avec un journaliste pour trouver ce qu'il recherche spécifiquement et comment nous pouvons lui donner les informations pertinentes du mieux possible.

J'explique comment les choses fonctionnent, principalement sur les processus ; cela met mes « vrais » porte-parole, ceux des médias, en meilleure position pour dire quels sont leurs besoins et minimise le risque de perdre leur participation en chemin. Préparer les bonnes personnes est aussi important que de les rendre disponibles. Pendant mes cours de formation aux médias, je mets quelques diapositives « surprenantes » qui soulignent les leçons particulièrement intéressantes apprises au fil des ans.

Nous avons par exemple connu une pagaille de représentants dans les premiers jours de l'incubateur Apache, où 15 personnes ont répondu à une demande de la presse en 48 heures… beaucoup d'opinions, mais qui était la « bonne » personne à citer ? Ne laissez pas la presse en décider ! Un autre scénario surprenant comprenait une fête de lancement globale avec des centaines d'invités, des représentants de la presse partout, des DJ, de la musique à fond, des cocktails à flot, et tout ça durerait jusqu'à très tard dans la nuit avec des rumeurs d'*after* pour continuer la fête.

Très tôt le matin suivant, la presse a débarqué (oui, bien sûr, j'accepte les appels du *Financial Times* à quatre heures du matin !). J'ai accepté avec excitation. Cependant, il s'avéra que nous n'avions pas de représentant disponible : le président était dans un avion à destination du Japon, le téléphone portable du directeur était éteint (avec une bonne raison, apparemment) ; les membres du conseil d'administration indisponibles, l'équipe non préparée. Des dizaines d'occasions manquées. Rappelez-vous : quand le communiqué de presse est diffusé, le travail commence tout juste.

Ne soyez pas surpris de les voir dire tout et n'importe quoi

Ils ont tous un avis. Et ils vont probablement vous le donner.

Ne compliquez pas les choses à outrance

Si vous pensez que vous avez trop de choses à dire… c'est probablement le cas. Les facultés d'attention ne sont plus ce qu'elles étaient ; être distrait et passer à autre chose est à portée de clic. Rappelez-vous que vous pouvez toujours travailler par étapes. Décomposez votre histoire si nécessaire. Coupez un long communiqué de presse et utilisez des supports documentaires comme des fiches de description technique et des pages de témoignages à la place. Le principe de segmentation (« cinq parties plus ou moins deux ») est quelque chose que j'utilise encore et encore. Créez votre propre cycle de publication pour vos messages et renforcez régulièrement votre présence. Créez une FAQ ; si une question mérite d'être posée et n'y est pas, trouvez le moyen de compléter votre message. La répétition engendre la familiarité. Le renforcement progressif de votre appel à l'action est une bonne chose.

N'y touchez plus pendant 24 heures

Parfois, vous avez besoin de prendre du champ. Vous éloigner d'un projet, d'un raisonnement, du travail en général. Accordez-vous une pause et essayez de garder un certain rythme. Prenez une journée pour laisser décanter et vous permettre de souffler. Bien que ce ne soit pas possible dans une entreprise gouvernée par les dates butoir, c'est un but à viser. La course effrénée, les courriels incessants et les tweets en continu déclenchent souvent des réactions à des urgences qui n'existent pas. Laissez le projet de côté, videz-vous la tête et revenez avec des idées claires. Faites un pas de côté et reprenez votre vie en main.

Visez haut

Placez haut la barre et soyez conscient de votre valeur.

Conférences et compagnie

34. L’important, c’est les gens

Nóirín Plunkett

Nóirín Plunkett est une touche-à-tout qui maîtrise plusieurs domaines. Rédactrice technique professionnelle, son travail open source *illustre l’expression « Si vous voulez que quelque chose soit fait, demandez à une personne occupée ». Nóirín a commencé dans l’*open source *avec Apache, donnant un coup de main sur la documentation du projet httpd. En moins d’un an, elle a été recrutée dans l’équipe de planification des conférences, qu’elle dirige désormais. Elle a participé à la mise en place du projet de développement communautaire chez Apache et a déjà agi en tant qu’administratrice d’organisation pour le Summer of Code[1]. Elle siège aux conseils d’administration de la fondation du logiciel Apache et de l’Initiative Open Cloud. Quand elle n’est pas en ligne, elle est dans son élément naturel sur une piste de danse. Mais c’est également une harpiste et chanteuse talentueuse et une excellente sous-chef[2].*

Il n’existe pas de chemin tout tracé, d’ailleurs celui que j’ai choisi était peut-être moins classique que la plupart. J’ai fait ma première contribution quand j’avais la vingtaine. À cette époque, j’avais déjà travaillé plus d’un an chez Microsoft. Mais après Microsoft, j’ai déménagé à l’étranger afin de poursuivre mes études et comme c’était sympa d’avoir un passe-temps à côté, j’ai commencé à travailler sur différentes documentations et traductions et j’ai contribué au projet *httpd* d’Apache.

1 http://fr.wikipedia.org/wiki/Google_Summer_of_Code.
2 [NdT] en français dans le texte.

Comme par hasard, bien sûr, la conférence européenne sur Apache allait avoir lieu à Dublin, alors que, cet été-là, j'étudiais à Munich. Mais la chance sourit aux Irlandais et, avec un peu d'astuce, j'ai convaincu Sun Microsystems de financer ma participation à la conférence.

J'ai une photo du moment précis où j'ai pris conscience que cette chose appelée *open source* était bien réelle, et que ça allait changer le monde. C'était pendant la soirée avant la conférence. Nous n'avions toujours pas trouvé où la fibre optique arrivait, elle était censée constituer la colonne vertébrale de notre réseau. Nous avions vérifié chaque coin, chaque armoire et chaque plinthe, en vain. Nous avions laissé tomber pour cette nuit, et nous étions occupés à nous assurer que les salles qui accueilleraient les sessions de formation auraient au moins suffisamment de connectivité pour que les formateurs puissent utiliser leurs supports de présentation[1].

Et à mesure que la nuit tombait, que les routeurs révélaient lentement les secrets de leur configuration par défaut, la demi-douzaine de volontaires, des gens que je n'avais rencontrés que dans l'après-midi même, devenaient des amis.

Je ne pourrais pas vous dire où sont les six filles avec lesquelles j'ai vécu pendant cet été-là à Munich. Mais je suis toujours en contact avec chacune des personnes que l'on peut voir sur cette photo. L'une d'elles a déménagé dans un autre pays, une autre est partie sur un autre continent. La plupart ont changé de travail entre-temps, j'ai eu mon diplôme et je me suis conformée à la grande tradition irlandaise de l'émigration pour trouver du travail.

Voyez-vous, l'*open source*, c'est d'abord des gens. Vraiment, sur presque n'importe quel projet dont vous voudriez faire partie, le code ne vient qu'après.

---

1 [NdA] Le lendemain matin, nous sommes allés dans les combles pour essayer de trouver la fibre, toujours rien. Pour finir, nous l'avons trouvée dans le local technique de la boîte de nuit, située dans le sous-sol à côté.

Ce qui fait que travailler sur un projet est un bonheur et non un calvaire, ce sont les gens. Ce qui fait qu'un projet prospère plutôt qu'il ne stagne, ce sont les gens. Bien entendu, vous serez capable de coder toute la nuit pour un projet si ça permet de résoudre un problème que vous pensez être important ; mais, à moins d'avoir des gens avec lesquels vous pouvez collaborer, discuter, concevoir et développer, vous allez probablement finir par perdre la motivation ou vous retrouver bloqué pour un bout de temps.

La vraie valeur des conférences, des sprints, les hackathons, des « retraites »[1] ou de tout ce que votre communauté appelle ses « moments de face à face », la voici : pouvoir retrouver face à face les personnes avec lesquelles vous avez travaillé. Les êtres humains sont des animaux sociaux ; les bébés reconnaissent des visages avant même de commencer à gazouiller, et peu importe à quel point les gens sont polis ou amicaux dans leurs courriels, il y a toujours quelque chose qui manque dans ces communications-là.

Rencontrer des gens en face à face nous donne une occasion de voir l'humanité de ceux avec qui on a pu avoir du mal à s'entendre, de partager la joie du travail bien fait avec ceux avec qui on aime travailler. Ainsi, si j'avais un conseil à donner aux débutants, et j'aurais aimé qu'on me le donne, ce serait de sortir, de rencontrer des gens, de coller des noms aux visages dès que l'occasion se présente[2].

---

1 [NdT] Une ou plusieurs journées qui se concentrent sur la création de code de très bonne qualité plutôt qu'écrit dans l'urgence.

2 [NdA] Malheureusement, je dois faire une sorte de mise en garde : comme pour tout rassemblement important, assister à une conférence *open source* présente des risques. Certains pires que d'autres, mais d'après mon expérience, les agressions, particulièrement, semblent plus fréquentes dans les communautés techniques que dans les communautés non techniques. Dénichez les évènements qui publient un code de conduite ou une politique anti-harcèlement et demandez de l'aide si vous ne vous sentez pas en sécurité. La grande majorité des gens que vous trouverez dans un événement *open source* sont des êtres humains formidables et attentionnés. J'espère qu'avec le temps, changer les attitudes empêchera la minorité de penser qu'elle peut se permettre des comportements déraisonnables dans ce genre de lieux…

Et si vous trouvez que les occasions sont rares et espacées, n'hésitez pas à demander. Cherchez des gens qui passent près de chez vous ou qui vivent là où vous voyagez, dénichez un parrainage pour assister aux grands évènements de la communauté, organisez votre propre événement !

C'est la richesse de nos communautés qui donne toute sa valeur à l'*open source*, ainsi que les efforts partagés vers des objectifs communs. Et bien sûr, les sessions musique, les repas, les pintes et les soirées ! Ce sont les choses qui nous rassemblent, et vous allez découvrir qu'une fois que vous aurez rencontré les gens en personne, même vos interactions par courriel seront plus riches, plus gratifiantes et plus fructueuses qu'auparavant.

35. Organiser des évènements

Dave Neary

Dave Neary travaille sur des projets libres et open source *depuis qu'il a découvert Linux en 1996. C'est un contributeur de GNOME et GIMP depuis longtemps. Il travaille à plein temps depuis 2007 pour réconcilier les entreprises avec les logiciels développés de façon communautaire. Durant cette période, il a travaillé sur divers projets dont OpenWengo, Maemo et Meego, qui étaient liés à de l'événementiel, à des méthodes de travail communautaires, à de la gestion de produits et d'outils d'analyse d'une communauté. Il s'est impliqué, en tant que bénévole, dans l'organisation du GUADEC, du Desktop Summit, du Libre Graphics Meeting, de la GIMP Conference, d'Ignite Lyon, de l'Open World Forum et de la MeeGo Conference.*

À part écrire du code, rassembler les principaux contributeurs aussi souvent que possible est l'une des choses les plus importantes que vous puissiez faire pour un projet lié au logiciel libre. J'ai eu la chance de pouvoir organiser un certain nombre d'évènements au cours de ces dix dernières années, mais aussi d'observer les autres et d'apprendre à leur contact pendant ce temps. Voici quelques-unes des leçons que j'ai apprises chemin faisant, grâce à cette expérience.

Le lieu

Le point de départ pour la plupart des réunions ou des conférences, c'est le lieu. Si vous réunissez un petit groupe (moins de dix personnes), il suffit bien souvent de choisir une ville et de demander à un ami qui possède une entreprise ou qui est professeur d'université de vous réserver une salle. Vous aurez sans doute besoin d'une organisation plus formelle dès qu'il y aura davantage de monde à accueillir.

Si vous ne faites pas attention, le lieu de l'événement sera une grande source de dépenses et il vous faudra trouver l'argent correspondant quelque part. Mais si vous êtes futé, vous pouvez vous débrouiller pour obtenir assez facilement une salle gratuitement.

Voici quelques-unes des stratégies qu'il serait intéressant d'essayer :

Intégrez-vous au sein d'un autre événement : le Linux Foundation Collaboration Summit, OSCON, LinuxTag, GUADEC et beaucoup d'autres conférences accueillent volontiers des ateliers ou des rencontres pour de plus petits groupes. Le GIMP Developers Conference en 2004 a été le premier rassemblement que j'ai organisé. Afin de ne pas devoir affronter les difficultés inhérentes au fait de trouver une salle, une date qui convienne à tout le monde et ainsi de suite, j'ai demandé à la GNOME Foundation si ça ne les dérangeait pas de nous laisser un peu d'espace lors de la GUADEC – et ils ont accepté. Tirez parti de l'organisation d'une conférence plus grande, et vous pourrez en plus y assister par la même occasion !

Demandez aux universités environnantes si elles n'ont pas des salles disponibles. Ça ne fonctionnera plus une fois que vous aurez dépassé une certaine taille, néanmoins, vous pouvez vous renseigner en particulier dans les universités dont certains chercheurs sont des membres du Linux User Group (LUG)[1] local. Ils

1 [NdT] groupe d'utilisateurs de Linux.

peuvent en parler avec leur directeur de département afin de réserver un amphithéâtre et quelques salles de classe pour un week-end. Un grand nombre d'universités vous demanderont de faire un communiqué de presse et d'être mentionnées sur le site Web de la conférence, ce qui est un juste retour des choses. Le premier Libre Graphics Meeting s'est déroulé gratuitement à CPE Lyon et le GNOME Boston Summit a été accueilli gratuitement pendant des années par le MIT.

Si le lieu de rendez-vous ne peut pas être gratuit, voyez si quelqu'un d'autre ne peut pas le financer. Lorsque votre conférence commence à accueillir plus de 200 personnes, la plupart des salles seront payantes. Héberger une conférence coûtera beaucoup d'argent à celui qui prête les locaux, et une part importante des ressources économiques des universités vient de l'organisation de conférences lorsque les étudiants sont partis. Mais si le centre de conférence ou l'université ne peut pas vous accueillir gratuitement, cela ne signifie pas ce soit à vous de payer. Les collectivités territoriales aiment bien être impliquées lorsqu'il s'agit d'organiser des évènements importants dans leur région. Le GUADEC à Stuttgart, le Gran Canaria Desktop Summit et le Desktop Summit à Berlin ont tous été financés par la région d'accueil en ce qui concerne la salle. S'associer avec une région présente un avantage supplémentaire : elles ont souvent des liens avec les entreprises et la presse locale, ce qui représente des ressources que vous pouvez utiliser afin d'obtenir de la visibilité et peut-être même des sponsors pour votre conférence.

Faites un appel d'offres : en invitant les institutions qui souhaitent héberger la conférence à déposer une offre, vous les incitez aussi à trouver une salle et à discuter avec les partenaires locaux avant de vous décider sur l'endroit où aller. Vous mettez aussi les villes en concurrence, et comme pour les candidatures aux Jeux Olympiques, les villes n'apprécient pas de perdre les compétitions auxquelles elles participent !

Le budget

Les conférences coûtent de l'argent. Ce qui peut coûter le plus cher pour une petite rencontre, ce sont les frais de déplacement des participants. Pour une conférence plus importante, les principaux coûts seront l'équipement, le personnel et la salle. Chaque fois que j'ai dû réunir un budget pour une conférence, mon approche globale a été simple :

- Décider du montant financier nécessaire pour réaliser l'événement.
- Collecter les fonds jusqu'à atteindre ce montant.
- Arrêter la collecte et passer aux étapes suivantes de l'organisation.

Lever des fonds est une chose difficile. On peut vraiment y passer tout son temps. Au bout du compte, il y a une conférence à préparer, et le montant du budget n'est pas la préoccupation principale de vos participants.

Rappelez-vous que votre objectif principal est de réunir les participants du projet afin de le faire avancer. Alors, obtenir des réponses de participants potentiels, organiser l'hébergement, prévoir la salle, les discours, la nourriture et la boisson, les activités festives et tout le reste que les participants attendent d'un événement… tout cela est plus important que la levée de fonds.

Bien sûr, de l'argent est nécessaire pour être capable d'organiser tout le reste. Alors, trouver des sponsors, décider de leurs niveaux de participation et vendre la conférence est un mal nécessaire. Mais une fois que vous avez atteint le montant requis pour la conférence, vous avez vraiment mieux à faire.

Il existe quelques sources potentielles de financement pour préparer une conférence et combiner ces sources me semble la meilleure façon d'augmenter vos recettes.

Les participants : même si c'est un sujet de controverse dans de nombreuses communautés, je crois qu'il est tout à fait justifié de demander aux participants de contribuer en partie aux coûts de la conférence. Les participants profitent des installations ainsi que des évènements sociaux et bénéficient de la conférence. Certaines communautés considèrent la participation à leur événement annuel comme une récompense pour services rendus ou comme une incitation à faire du bon travail dans l'année à venir. Mais je ne crois pas que ce soit une façon pérenne de voir les choses. Pour les participants à une conférence, il existe plusieurs façons de financer l'organisation :

1. *Les droits d'inscription :* c'est la méthode la plus courante pour obtenir de l'argent de la part des participants à la conférence. La plupart des conférences communautaires demandent un montant symbolique. J'ai vu des conférences qui demandaient un droit d'entrée de 20 à 50 euros ; et ça ne posait aucun problème à la plupart des gens de payer cela. Un règlement d'avance a l'avantage supplémentaire de réduire massivement les désistements parmi les gens qui vivent à proximité. Les gens accordent plus d'importance à la participation à un événement leur coûtant dix euros qu'à un autre dont l'entrée est gratuite, même si le contenu est identique.

2. *Les dons :* ils sont utilisés avec beaucoup de succès par le FOSDEM. Les participants reçoivent un ensemble de petits cadeaux fournis par les sponsors (livres, abonnements à des magazines, T-shirts) en échange d'un don. Mais ceux qui le souhaitent peuvent venir gratuitement.

3. *La vente de produits dérivés :* votre communauté serait peut-être plus heureuse d'accueillir une conférence gratuite et de vendre des peluches, des T-shirts, des sweats à capuche, des mugs et d'autres produits dérivés afin de récolter de l'argent. Attention ! D'après mon expérience, on peut s'attendre à obtenir moins de bénéfices de la vente de

produits dérivés qu’on n’en obtiendrait en offrant un T-shirt à chaque participant ayant payé un droit d’inscription.

Les sponsors : les publications dans les médias accepteront généralement un « partenariat de presse » – en faisant de la publicité pour votre conférence dans leur magazine papier ou sur leur site Web. Si votre conférence est déclarée comme émanant d’une association à but non lucratif pouvant accepter des dons avec des déductions d’impôts, proposez à vos partenaires dans la presse de vous facturer pour les services et de vous donner ensuite une subvention de partenariat séparée pour couvrir la facture. Le résultat final est identique pour vous. Mais il permettra à la publication de compenser l’espace qu’elle vous vend par des réductions d’impôts. Ce que vous souhaitez vraiment, ce sont des parrainages en liquide. Comme le nombre de projets de logiciels libres et les conférences se sont multipliés ces dernières années, la compétition pour les parrainages en liquide s’est intensifiée. Afin de maximiser vos chances d’atteindre le budget que vous vous êtes fixé, voici ce que vous pouvez entreprendre :

1. *Une brochure de la conférence :* pensez à votre conférence comme à un produit que vous vendez. Que représente-t-elle, quelle attention attire-t-elle, à quel point est-elle importante pour vous, pour vos membres, pour l’industrie et au-delà ? Qu’est-ce qui a de la valeur pour votre sponsor ? Vous pouvez vendre un contrat de parrainage sur trois ou quatre éléments différents : peut-être que les participants à votre conférence constituent un public cible de grande valeur pour le sponsor, peut-être (en particulier pour les conférences de moindre importance) que les participants ne sont pas ce qui est important mais plutôt la couverture dont bénéficiera la conférence dans la presse internationale ou bien peut-être vendrez-vous à l’entreprise le fait que la conférence améliore un logiciel dont elle dépend. En fonction du positionnement de la confé-

rence, vous pouvez lister les sponsors potentiels. Vous devriez avoir une brochure de parrainage que vous pourrez leur envoyer. Elle devra contenir une description de la conférence, un argumentaire de vente expliquant pourquoi il est intéressant pour l'entreprise de la parrainer, éventuellement des coupures de presse ou des citations de participants à des éditions antérieures disant à quel point votre conférence est géniale et, finalement, la somme d'argent que vous recherchez.

2. *Des niveaux de parrainage :* ils devraient être fixés en fonction de la somme que vous voulez lever. Vous devriez attendre de votre sponsor le plus important qu'il vous fournisse entre 30 et 40 % du budget total de la conférence, pour une conférence de moindre importance. Si vous êtes chanceux et que votre conférence attire de nombreux sponsors, cela peut s'élever à seulement 20 %. Pour vos estimations, visez un tiers. Ceci signifie que si vous avez décidé que vous avez besoin de 60 000 euros, vous devriez alors mettre votre niveau de sponsor principal à 20 000 euros et tous les autres niveaux en conséquence (disons 12 000 euros pour le deuxième niveau et 6 000 pour le troisième). Pour les conférences de moindre importance et les rencontres, le processus peut être légèrement plus informel. Mais vous devriez toujours penser au processus entier comme un argumentaire de vente.

3. *Un calendrier :* la plupart des entreprises ont un cycle budgétaire soit annuel, soit semestriel. Si vous envoyez votre demande à la bonne personne au bon moment, vous pourriez alors avoir une discussion bien plus aisée. Le meilleur moment pour soumettre des propositions de parrainage d'une conférence estivale est situé aux environs d'octobre ou de novembre de l'année précédente, lorsque les entreprises finalisent leur budget annuel. Si vous manquez cette fenêtre, tout n'est pas perdu. Mais tout

parrainage que vous obtiendrez viendra des budgets de fonctionnement qui tendent à être maigres et qui sont gardés précieusement par leurs propriétaires. Sinon, vous pouvez obtenir un engagement de parrainage en mai pour votre conférence de juin, à la fin du processus budgétaire du premier semestre – ce qui est tardif dans la préparation.

4. *Les bonnes personnes :* je ne vais pas enseigner l'art de la vente à qui que ce soit mais mon secret personnel dans les négociations avec les grandes organisations est de devenir ami avec des personnes à l'intérieur de ces organisations et de me forger une impression sur l'origine potentielle du budget pour mon événement. Votre ami ne sera probablement pas la personne qui contrôle le budget mais l'avoir à vos côtés est une chance d'avoir un allié au sein de l'organisation. Il fera en sorte que votre proposition soit mise devant les yeux de la personne en charge du budget. Les grandes organisations peuvent être aussi dures qu'une noix est dure à craquer, mais les projets de logiciels libres ont souvent des amis dans les hautes sphères. Si vous avez vu le directeur technique ou le PDG d'une entreprise classée au Fortune 500 parler de votre projet dans un article de journal, n'hésitez pas à lui envoyer quelques mots de remerciements, et quand le temps sera venu de financer cette conférence, une note personnelle demandant qui est la meilleure personne à contacter fera des merveilles. Souvenez-vous que votre objectif n'est pas de vendre quoi que ce soit à votre contact personnel mais de le changer en un défenseur de votre cause à l'intérieur de l'organisation et de vous créer la possibilité de vendre par la suite la conférence à la personne responsable du budget.

Souvenez-vous aussi, en vendant des contrats de parrainage, que tout ce qui vous coûte de l'argent pourrait en faire partie. Certaines entreprises offriront des tours de cou aux participants, la pause café, la glace de l'après-midi ou bien un événement social.

Ce sont de bonnes occasions de parrainage et vous devriez exprimer clairement, dans votre brochure, tout ce qui se déroule. Vous devriez aussi définir un budget prévisionnel pour chacun de ces évènements lorsque vous écrivez le brouillon de votre budget.

Le contenu

Le contenu d'une conférence est son élément le plus important. Des évènements différents peuvent traiter différemment d'un même contenu – certains évènements invitent une grande partie de leurs intervenants, tandis que d'autres comme GUADEC et OSCON font des appels à propositions et choisissent les interventions qui rempliront les salles.

La stratégie que vous choisirez dépendra beaucoup de la nature de l'événement. Si l'événement existe depuis une dizaine d'années, avec un nombre de participants toujours croissant, faire un appel à articles est une bonne idée. Si vous êtes dans votre première année, et si les personnes ne savent vraiment pas quoi faire de l'événement, alors donnez le ton en invitant de nombreux orateurs, ainsi les gens comprendront votre objectif.

Pour Ignite Lyon l'an dernier, j'ai invité environ 40 % des orateurs pour le premier soir (et j'ai souvent dû les harceler pour qu'ils me proposent une intervention). Les 60 % restants sont venus via un formulaire de candidature. Pour le premier Libre Graphics Meeting, en dehors des présentations éclair, je pense avoir d'abord contacté chaque orateur, à l'exception de deux d'entre eux. Maintenant que l'événement en est à sa sixième année, il existe un processus d'appel à contributions qui fonctionne plutôt bien.

Le programme

Il est difficile d'éviter de mettre en parallèle des exposés attrayants pour les mêmes personnes. Dans chaque conférence, vous pouvez entendre des personnes qui voulaient assister à des

conférences se déroulant en même temps, sur des sujets similaires.

Ma solution pour la programmation des conférences est très simple, mais elle fonctionne dans mon cas. Des post-it de couleur, avec une couleur différente pour chaque thème, et une grille vide. Le tour est joué. Écrivez les titres des exposés (un par post-it), ajoutez les quelques contraintes que vous avez pour l'orateur, puis remplissez la grille.

Mettre le programme en dehors de l'ordinateur, et dans des objets réels, permet de voir très facilement les conflits, d'échanger les conférences aussi souvent que vous voulez, et de le publier ensuite sur une page Web quand vous en êtes satisfait.

J'ai utilisé cette technique avec succès pour le GUADEC 2006[1] et Ross Burton[2] l'a réutilisée avec succès en 2007.

Les fêtes

Les fêtes sont un compromis à trouver. Vous voulez que tout le monde s'amuse, et s'éclater jusqu'au petit matin est une motivation importante pour participer à une conférence. Mais la fréquentation matinale souffre après une fête. Ayez pitié du pauvre membre de la communauté qui doit se tirer du lit après trois heures de sommeil pour aller parler devant quatre personnes à 9 heures du matin après la fête.

Certaines conférences ont trop de fêtes. C'est super d'avoir la possibilité de se saouler avec des amis chaque nuit. Mais ça n'est pas génial de vraiment le faire chaque nuit. Rappelez-vous du but de la conférence : vous voulez que votre projet fasse un pas en avant.

Je préconise une grande fête, et une plus petite, au cours de la semaine. En dehors de ça, les personnes seront tout de même

1 http://blogs.gnome.org/bolsh/2006/05/09/initial-schedule-ready.
2 http://www.flickr.com/photos/rossburton/467140094.

ensemble et passeront de bons moments, mais ce sera à leurs frais, ce qui fait que chacun restera raisonnable.

Avec un peu d'imagination, vous pouvez organiser des évènements qui n'impliquent pas l'utilisation de musique forte et d'alcool. D'autres genres d'évènements sociaux peuvent faire l'affaire, et même être plus amusants.

Au GUADEC, nous organisons un tournoi de football depuis quelques années. Lors du sommet OpenWengo en 2007, nous avions embarqué les personnes pour une balade en bateau sur la Seine puis nous étions ensuite montés sur un manège du XIXe siècle. Faire manger les gens ensemble est un autre moyen de nouer des liens. J'ai de très agréables souvenirs de repas de groupe lors de nombreuses conférences. À la conférence annuelle de KDE, l'Akademy, il y a traditionnellement une grande journée de sortie durant laquelle les personnes vont ensemble à un pique-nique, quelques activités simples de plein air, une promenade en bateau, un peu de tourisme ou quelque chose de similaire.

Les coûts supplémentaires

Attention à ces dépenses imprévues ! Une conférence dans laquelle j'étais impliqué, et où le lieu de réunion était « sponsorisé à 100 % » nous a laissé une note de 20 000 euros pour les coûts de main-d'œuvre et d'équipement. Oui, le lieu était sponsorisé, mais la mise en place des tables et des chaises, ainsi que la location des écrans, des vidéoprojecteurs et de tout le reste ne l'était pas. Au bout du compte, j'ai estimé que nous avions utilisé seulement 60 % de l'équipement que nous avions payé.

Tout ce qui est fourni sur place est extrêmement coûteux. Une pause-café peut coûter jusqu'à 8 euros par personne pour un café et quelques biscuits, de l'eau en bouteille pour les conférenciers coûte quatre euros par bouteille, etc. La location d'un rétroprojecteur et de micros pour une salle peut coûter 300 euros ou plus pour une journée, selon que le propriétaire exige ou non que

l'équipement audio-vidéo soit manipulé par son propre technicien.

Quand vous traitez avec un lieu commercial, soyez clair dès le départ sur ce pour quoi vous payez.

Les détails sur le site

J'aime les conférences attentives aux petits détails. En tant qu'orateur, j'aime quand quelqu'un me contacte avant la conférence pour m'avertir qu'il me présentera et me demande ce que je souhaiterais qu'il dise. C'est rassurant aussi de savoir que, quand j'arriverai, il y aura un micro sans fil et quelqu'un qui peut aider à l'ajuster.

Faire attention à tous ces détails nécessite de nombreux volontaires et quelqu'un pour tout ré avant et pendant l'événement. Il faut passer beaucoup de temps à parler à l'équipe sur place, plus particulièrement aux techniciens audio-vidéo.

Lors d'une conférence, le technicien audio-vidéo avait prévu de basculer manuellement l'affichage vers un économiseur d'écran à la fin d'une conférence. Au cours d'une session de mini-conférences, nous nous sommes retrouvés dans une situation burlesque quand, après le premier conférencier, j'ai interverti l'ordre de passage : au moment où la présentation suivante s'affichait sur mon portable, nous avions toujours l'économiseur sur le grand écran. Personne n'avait parlé avec le technicien de la régie pour lui expliquer le format de la présentation ! Et c'est comme ça qu'on a fini par avoir pas moins de quatre spécialistes de Linux à s'occuper de l'ordinateur portable, qui vérifiaient les connexions en psalmodiant divers mantras Xrandr, qui s'efforçaient de remettre en marche le rétroprojecteur au-dessus de nos têtes ! Nous avons fini par changer d'ordinateur portable, le technicien de la régie a compris de quel type de session il s'agissait, et ensuite tout s'est fort bien passé – la plupart des gens concernés ont accusé mon portable.

Gérer une conférence, ou parfois une plus petite rencontre, prend du temps et nécessite beaucoup d'attention aux détails, qui pour la plupart ne seront jamais remarqués par les participants. Et je n'ai même pas évoqué des choses comme les banderoles et les affiches, la création du graphisme, la gestion de la presse ou d'autres joyeusetés qui vont de pair avec l'organisation d'une conférence.

Le résultat final est en revanche particulièrement gratifiant. Une étude que j'ai menée l'année dernière sur le projet GNOME a montré qu'il y a eu une forte augmentation de la productivité sur tout le projet juste après notre conférence annuelle et un grand nombre de membres de notre communauté mentionnent la conférence comme ayant été, pour eux, le point culminant de l'année.

36. Gérer les conférences

Gareth J. Greenaway

Gareth J. Greenaway s'est impliqué activement dans la communauté du logiciel libre et open source *depuis 1997 après avoir découvert Linux. Sa contribution majeure a consisté à regrouper des gens ayant la même opinion pour apprendre et expérimenter de nouveaux éléments de logiciel libre et* open source. *Cette implication a débuté avec un petit groupe d'utilisateurs de Linux (un « GUL ») et s'est développée avec l'organisation de la « SouthernCalifornia Linux Expo », aussi connue sous le nom de SCALE. En tant que membre fondateur de cet événement, Gareth remplit actuellement deux fonctions importantes au sein de l'organisation. La première est la gestion des conférences, la seconde concerne les relations avec la communauté.*

J'ai commencé à écrire cette section avec ce que je pensais être les besoins et les étapes pour organiser une conférence sur le libre et l'*open source*. Cependant, une grande partie de ce que je trouvais à dire avait déjà été abordée par Dave Neary, expert en gestion de communautés. Donc, pour éviter de répéter et de recouper ce que Dave voulait expliquer, j'ai décidé de partager différentes histoires de l'organisation de SCALE[1] et les leçons que j'en ai tirées pendant ces années.

1 https://www.socallinuxexpo.org/scale11x.

Trop d’énergie !

SCALE a commencé il y a maintenant neuf ans avec des membres de trois groupes locaux d’utilisateurs de Linux : ce n’était à l’origine qu’un modeste événement régional organisé par l’un de ces groupes. La première expérience fut vraiment enrichissante. Beaucoup de leçons en ont été tirées. On courait un peu partout et l’événement semblait se dérouler à une vitesse folle. Étant donné qu’aucun de nous n’avait encore organisé d’événement où il fallait se soucier des risques de surtension ou de consommation électrique, nous n’y avons pas pensé, et, du coup, nous avons dû ré-enclencher les disjoncteurs de la salle plusieurs fois pendant l’événement.

Connexion sans fil et sans filet

Pour le deuxième SCALE nous avons retenu les leçons apprises l’année précédente mais un nouveau lieu de rencontre allait infliger de nouvelles leçons. Le centre de conférences de Los Angeles est le lieu où s’est tenu SCALE 2, il fournissait un espace bien plus grand pour installer l’événement. Le nouveau lieu nous a aussi permis d’apprendre notre première leçon sur les contrats avec un grand organisme pour gérer les choses telles que l’équipement audio et vidéo, l’accès à Internet et le matériel d’exposition.

Compte tenu de la situation de l’événement à l’intérieur du centre de conférences, nous avons dû positionner les comptoirs d’enregistrement dans une zone qui, tout en étant visible des participants qui arrivaient, se trouvait à une certaine distance du reste du salon. Nos possibilités pour fournir un accès réseau à la zone d’enregistrement étaient limitées car les règles de protection anti-incendie proscrivaient l’utilisation de câbles ; le sans-fil était donc l’unique possibilité.

Tout a été mis en place très tôt le jour du salon et fonctionnait parfaitement bien, jusqu’à ce que cela cesse mystérieusement de

marcher. La connexion sans fil qui fournissait l’accès au réseau indispensable au comptoir d’enregistrement a tout simplement disparu. Nous avons alors connu beaucoup de tentatives de dépannages, beaucoup de déplacements d’équipements et d’antennes et beaucoup de frustration. « Ça devrait fonctionner », telle était la seule conclusion à laquelle tout le monde parvenait, sans trop savoir pourquoi cela ne fonctionnait pas.

Soudain, un des membres de l’équipe qui s’était tenu à l’écart de la séance de dépannage a appelé tout le monde à le rejoindre là où il se trouvait. En face d’une grande fenêtre qui surplombait un grand hall de réunion, nous avons tout à coup vu ce qu’il désirait nous faire voir. En dessous de nous, il y avait des dizaines de lumières clignotantes, tournantes et pulsantes qui nous regardaient. Des centaines d’appareils électroniques avec des lumières clignotantes, des sirènes et des panneaux à LED (diodes électroluminescentes), interférant narquoisement avec les signaux sans fil de nos pauvres points d’accès. Nous avons soudain compris que nos heures de travail à tenter de résoudre ce problème de sans-fil avaient été vaines. Finalement, nous avons déroulé un câble Ethernet, l’avons scotché en faisant de notre mieux pour que tout soit en sécurité, et nous avons dit une petite prière pour que le capitaine des pompiers ne fasse pas d’inspection surprise.

Soirées de gala, tireurs d’élite et l’affaire de la mallette IBM disparue

L’une des anecdotes les plus célèbres dans l’histoire de SCALE est sans doute celle des incidents et péripéties qui sont survenus pendant SCALE 3. Ces tribulations sont bien connues, et si vous assistiez à SCALE cette année-là, vous n’avez pas pu y échapper.

Le troisième SCALE devait se dérouler encore une fois au *L.A. Convention Center*. Tout le travail de planification et d’organisation avait été mené en amont pendant de nombreux mois et tout s’annonçait bien. Trois semaines environ avant l’événement, nous avons reçu des informations à propos de plusieurs routes qui

seraient fermées autour du centre de conférences à cause d’une soirée de gala qui devait avoir lieu. À cause de ces fermetures de routes, il n’y avait plus qu’une voie d’accès pour accéder au centre et en repartir, ce qui est loin d’être idéal. Fort heureusement, nous avons eu le temps d’avertir tous ceux qui venaient à l’événement et de leur indiquer les routes fermées à la circulation et les itinéraires alternatifs. Cette année-là, c’était aussi la première fois que SCALE devait se dérouler sur deux jours, dans l’espoir de répartir un peu les choses pour ne pas être autant dans la précipitation et la frénésie.

IBM est l’un des plus anciens sponsors et exposants de SCALE. L’entreprise a toujours été une plus-value appréciée, mais, malheureusement, sa participation s’est généralement accompagnée de quelques difficultés. La veille de l’événement, comme d’habitude, avait été réservée à la mise en place pour permettre à l’équipe de SCALE de tout installer et aux exposants de préparer leurs stands. C’est également le jour de réception de tous les paquets envoyés par les exposants. IBM avait prévu de présenter une nouvelle ligne de serveurs sur le salon et avait fait expédier un de ces serveurs au centre de conférences ; malheureusement, il n’avait pas été livré sur leur stand et personne dans le centre de conférences ne savait où pouvait bien se trouver le colis. Malgré de nombreuses heures à chercher dans tous les endroits possibles à l’intérieur du centre de conférences, nous n’avions pas la moindre piste.

Il se trouve que pour le gala qui devait avoir lieu quelques jours plus tard, un certain nombre de pièces avaient été louées pour en faire des bureaux et des espaces de stockage. Dans un éclair de génie, le coordinateur de l’événement qui aidait à la recherche suggéra que nous pourrions chercher dans un de leurs espaces de stockage en espérant que la mallette d’IBM ait été livrée là par accident. La pièce en question était un petit placard de rangement dans lequel nous nous sommes trouvés face à des montagnes de boîtes, du sol jusqu’au plafond, remplies de tickets pour la soirée

de gala à venir. Derrière ces boîtes, dans un coin, il y avait une grande mallette bleue avec le logo IBM bien visible. Crise évitée !

Le reste de l'événement se déroula sans heurts et pratiquement sans incidents. Alors que la conférence se finissait, une petite foule commença à se former près de grandes fenêtres donnant sur la rue. Alors que je passais à cet endroit, je compris ce que tout le monde était en train de regarder. Plusieurs silhouettes, toutes vêtues de noir, se déplaçaient sur les toits des bâtiments le long de la rue. Toutes ces silhouettes portaient des fusils de précision et étaient des membres de l'équipe du SWAT de la police de Los Angeles qui se préparaient pour la soirée de gala prévue quelques heures plus tard. C'est dans une ambiance bien particulière que nous avons quitté le centre de conférences.

Pas de chambre à l'hôtel

Le quatrième SCALE a occasionné un nouveau changement de lieu. Cette fois-là, nous avons choisi un hôtel au lieu d'un centre de conférences. Comme avec les années, de plus en plus de personnes voyageaient pour assister à SCALE et séjournaient dans des hôtels proches, nous avons décidé d'étudier la possibilité que SCALE ait lieu dans un hôtel. Nous avons parcouru la région et avons fini par consulter un organisateur d'évènements pour trouver le bon endroit pour la conférence. Après nous être décidés pour un hôtel proche de l'aéroport de Los Angeles, la planification commença. Tenir l'événement dans un hôtel nous a rapidement confrontés à de nouvelles problématiques propres aux hôtels. Une des plus importantes leçons que nous avons alors apprises : toujours s'assurer que tous les contrats comportent une clause conventionnelle d'annulation.

À peu près cinq semaines avant le jour J, nous avons reçu un appel des responsables du lieu qui nous informaient que leur entreprise annulait notre événement pour attribuer le lieu à une autre manifestation. Cela a évidemment été un choc pour nous et

nous a plongés dans la plus grande confusion. Le contrat avec l'hôtel ne comprenait pas une ligne sur les cas de résiliation pour changement de lieu, mais précisait seulement qu'ils pouvaient annuler la manifestation sans aucun motif.

Après un grand nombre de coups de téléphone et de tractations avec les responsables du lieu initial, ils ont fini par accepter de nous indemniser pour nous aider à migrer vers un lieu de remplacement. Lequel nous a consenti les mêmes conditions pour tout ce qui concernait l'électricité, l'accès à Internet et l'équipement audio et vidéo. Tout s'est bien passé et l'équipe de SCALE en a tiré une précieuse leçon sur la façon de négocier ses futurs contrats.

Rappel !

Tout compte fait, organiser une conférence est une entreprise gratifiante et un excellent moyen de rendre à la communauté ce qu'elle nous a apporté. Les conférences constituent un moment privilégié car elles permettent des échanges en personne dans un monde qui repose couramment sur des moyens de communication virtuels.

Voici les conseils que je donnerais à des organisateurs de conférences :

- Commencez modestement, ne vous lancez pas dans un gigantesque événement dès la première année.
- Saisissez les occasions, faites des erreurs, n'ayez pas peur de l'échec.
- Tout est dans la communication !

37. Trouver les fonds

Selena Deckelmann

Selena Deckelmann est une importante contributrice de PostgreSQL[1]*. Elle donne des conférences dans le monde entier sur les logiciels libres, les communautés de développeurs et leurs trolls. Elle s'intéresse à l'ouverture des données publiques de la ville de Portland, aux* poulets d'appartement[2] *et à la recherche de solutions pour permettre aux bases de données de fonctionner plus vite. Elle a fondé Postgres Open, une conférence dédiée aux activités économiques autour de PostgreSQL et au bouleversement du secteur des bases de données. Elle a fondé et co-présidé Open Source Bridge, une conférence de développeurs pour les citoyens* open source. *Elle a fondé la Conférence PostgreSQL, une brillante série de conférences sur tous les États-Unis d'est en ouest. Elle fait actuellement partie du comité de programme de PgCon, de la conférence des utilisateurs MySQL et de OSCON Data. Elle est l'une des contributrices au manuel du mentor des Google Summer of Code, et du Guide des Étudiants. Elle est conseillère pour l'initiative Ada et membre du conseil de la société Technocation.*

Si je retrace mon parcours depuis la première fois où j'ai démarré un PC sous Linux en 1994, une chose ressort clairement de mon expérience avec l'*open source* : j'aurais aimé savoir comment demander de l'argent. Demander de l'argent est difficile. J'ai écrit des demandes de subventions, demandé des augmentations, négocié des salaires et des tarifs horaires de

1 https://fr.wikipedia.org/wiki/PostgreSQL.
2 http://urbanchickens.org.

consultante et levé des fonds pour des conférences à but non lucratif. Après de nombreuses tentatives et échecs, j’ai développé une méthode qui fonctionne ! Ce qui suit est un condensé des trucs et astuces que j’ai utilisés durant ces cinq dernières années pour augmenter les fonds pour des non-conférences, des sprints de code d’une journée et des conférences de plusieurs jours à propos de la culture et des logiciels *open source*.

La méthode pour obtenir des fonds pour une conférence comporte six étapes :

1. Identifier un besoin.
2. En parler à quelqu’un.
3. Demander de l’argent.
4. Récupérer l’argent.
5. Dépenser l’argent.
6. Remercier.

Identifier un besoin

Votre première mission en tant qu’organisateur de conférences consiste à expliquer pourquoi vous mettez en place une conférence de plus, en quoi elle sera utile à ceux qui y assisteront et quel intérêt un sponsor aurait à vous financer. On appelle ça rédiger un dossier de présentation ». Les éléments principaux d’un tel dossier sont les suivants :

L’objectif : en un paragraphe, expliquez pourquoi vous faites la conférence. Qu’est-ce qui vous a poussé à rassembler des gens ? Qui seront les participants ? De quoi parleront-ils une fois là-bas ? Si vous avez un sujet ou un but particulier en tête, mentionnez-le. Expliquez également pourquoi vous avez choisi tel endroit pour l’événement. Y a-t-il un lien avec le sujet de la conférence ? Est-ce que des personnalités intéressantes s’y trouveront ? Est-ce qu’il y a un sponsor ? Enfin, mettez à disposition

des chiffres significatifs à propos des évènements précédents, comme le nombre de participants et des informations pertinentes sur les intervenants ou des détails sur le lieu choisi.

Les possibilités de mécénat et les bénéfices escomptés : cette partie du dossier va mettre en relief ce que les sponsors peuvent attendre de votre conférence. En règle générale, on y expose les retours évalués en termes financiers, mais on peut également y décrire des avantages comme des travaux en nature ou du bénévolat. Commencez simplement. Traditionnellement, les parrainages financiers des évènements sont assurés par des services des ressources humaines qui cherchent à embaucher ou par des départements *promotion-marketing* qui cherchent à faire connaître leurs produits ou services. Voici, entre autres, le genre d'avantages que les sponsors en attendent : la mention du sponsor sur un site Web, dans les messages ou tweets pour les participants, l'accès à la liste des adresses électroniques ou aux informations sur les profils des participants, la présence des logos et des étiquettes sur les pochettes, tours de cou et autres gadgets distribués lors de la conférence, de même au moment des pauses café, des repas et casse-croûtes. Il leur faut aussi un stand sur la zone de la conférence et de l'espace publicitaire sur le programme de la conférence. Pensez aussi aux choses originales qui permettront de vous démarquer, à travers le déroulement et le lieu de la conférence. Par exemple, à Portland, il y a une boutique de beignets très populaire, avec un service de livraison. Nous avons trouvé un sponsor, et nous avons obtenu la permission d'amener le camion de livraison juste à l'endroit où nous étions et nous avons servi des beignets gratuitement pour le petit-déjeuner. Vous trouverez ci-dessous des liens pour des exemples de dossiers. Ils correspondent tous à de grosses conférences, donc vous n'obtiendrez peut-être pas le même résultat. J'ai déjà fait un dossier, avec une seule possibilité de parrainage, et l'accord était qu'en échange de la présence d'un de ses employés à la conférence, les organisateurs mentionnaient clairement l'entreprise et la remerciaient pour son soutien.

- OSCON : http://bit.ly/zd62Q6
- Open Source Bridge : http://bit.ly/dKWvYJ
- MeeGo San Francisco : http://bit.ly/zLUKEN

Le contrat : toujours inclure un contrat avec votre dossier. Il établit les attentes et les engagements ainsi que le calendrier et peut éviter beaucoup de problèmes en chemin. Je ne suis pas avocate, donc ce qui suit relève plus de mon expérience que des conseils juridiques. Pour les évènements plus mineurs, j'écris un contrat très simple qui expose mes attentes : les sponsors promettent de payer à une certaine date et je promets de tenir l'événement à une certaine date. Copier un contrat existant est quelque chose de délicat car les lois changent suivant les différents états et pays. J'ai consulté un avocat qu'un gestionnaire chevronné d'une communauté *open source* m'avait recommandé. Le cabinet d'avocats a été assez agréable pour gracieusement créer des contrats et réviser des contrats entre nous et les hôtels. Le Software Freedom Law Center peut vous indiquer un avocat approprié si vous n'en avez pas.

Maintenant que vous avez créé le dossier de présentation, vous devez en parler autour de vous.

En parler

L'étape la plus difficile pour moi, c'est de faire passer le mot au sujet de mes évènements ! Entraînez-vous à présenter votre événement en une ou deux phrases. Transmettez ce qui vous emballe et ce qui devrait emballer les autres.

Au fil des ans, j'ai appris qu'il fallait que je commence à parler SANS DÉLAI à mes connaissances plutôt que de m'inquiéter de savoir exactement quelles étaient les bonnes personnes à qui parler. Faites une liste des personnes à qui parler et que vous connaissez déjà et commencez à cocher cette liste.

La meilleure manière de parler de votre projet est de le faire en personne ou au téléphone. Ainsi, vous ne spammez pas les gens, vous captez leur attention et vous pouvez avoir un retour immédiat sur votre argumentaire. Les gens sont-ils enthousiastes ? Posent-ils des questions ? Ou bien trouvent-ils que c'est rasoir ? À qui d'autre pensent-ils que vous devriez en parler ? Demandez-leur ce qu'ils en pensent et comment vous pourriez rendre votre argumentaire plus attractif, plus intéressant, de sorte qu'ils en aient pour leur argent !

Une fois que vous aurez trouvé les mots-clés de votre argumentaire, écrivez-le et envoyez quelques courriels. Demandez des retours sur votre courriel et terminez toujours par un appel à agir avec une échéance pour la réponse. Gardez trace des personnes qui répondent, de leurs réponses et du moment favorable pour une relance de chacune d'elles.

Demander de l'argent

Armé de votre dossier et de votre argumentaire réglé aux petits oignons, commencez à approcher des entreprises pour financer votre événement. À chaque fois que je lance une nouvelle conférence, je fais une liste de questions à son propos et je réponds à chacune avec une liste de personnes et d'entreprises :

- Parmi les personnes que je connais, qui va trouver que c'est une idée géniale et faire la promo de mon événement (les « supporters »).
- Quelles sont les personnes dont la présence à la conférence serait vraiment sympa (les « experts reconnus »).
- Quelles sociétés ont des produits qu'elles voudraient promouvoir à mon événement (le « marketing »).
- Qui voudrait embaucher les personnes qui participent (les « recruteurs »).
- Quels projets libres et *open source* voudraient recruter des développeurs (les « recruteurs *open source* »).

En utilisant ces listes, envoyez votre brochure à travers le monde ! Voici un aperçu de la façon dont j'organise la démarche de sollicitation : je commence par envoyer les dossiers de présentation à mes supporters. J'en glisse aussi une copie aux experts, et je les invite à assister à la conférence ou à y intervenir. Je contacte ensuite les agences de publicité, les recruteurs et les recruteurs *open source* (parfois ça se recoupe !). En parallèle, j'ai généralement ouvert les inscriptions à la conférence et annoncé quelques allocutions ou évènements spéciaux. Je croise les doigts pour que ça pousse à quelques inscriptions, que ça aide les sponsors à sentir que cette conférence va certainement avoir lieu et que tout va bien se passer.

Récupérer l'argent

Si tout se passe comme prévu, des entreprises et des particuliers vont commencer à vous proposer de l'argent. Lorsque cela se produit, vous aurez besoin de deux choses très importantes :

- Un modèle de factures ou de devis.
- Un compte en banque pour recueillir les fonds.

Les modèles de factures sont simples à réaliser. J'utilise une feuille de calcul Google que j'actualise pour chaque facture. Vous pourriez facilement utiliser OpenOffice.org ou même TeX (si quelqu'un peut m'envoyer un modèle de facture en LaTeX, merci d'avance !). On peut trouver des exemples de factures à l'adresse http://www.freetemplatesdepot.com.

Les éléments les plus importants d'une facture sont : le mot FACTURE, un numéro unique de facture, le nom et les informations de contact du sponsor, le montant que le sponsor est censé verser, les termes de l'accord (à quelle date le sponsor est censé payer, quelles sont les pénalités en cas de non-paiement) et le montant total dû. Il faut ensuite envoyer une copie de la facture à la société. Gardez une copie pour vous !

Certaines sociétés peuvent exiger que vous remplissiez des formulaires plus ou moins complexes pour vous reconnaître, vous ou votre organisation, comme un fournisseur. De la paperasserie. Beurk ! Les délais de paiement pour de grandes entreprises peuvent atteindre deux mois. Les exercices budgétaires des sociétés sont en général annuels. Regardez si une société a un budget disponible pour votre événement et essayez d'être inclus dans les prévisions budgétaires de l'année suivante, si vous avez manqué l'occasion pour l'année en cours.

Le compte en banque peut être votre compte personnel, mais c'est risqué pour vous. Pour un événement à plusieurs milliers d'euros, vous préférerez peut-être trouver une ONG ou une association loi de 1901 qui peut détenir et dépenser les fonds en votre nom. Si votre conférence est à but lucratif, vous devriez consulter un comptable sur la meilleure manière de gérer ces fonds. Trouver une organisation sans but lucratif avec laquelle travailler peut se résumer à contacter une fondation qui gère un projet de logiciel libre.

Maintenant, passons à ce qui justifie tout ce processus : dépenser les dons durement acquis !

Dépenser l'argent

Maintenant que vos sponsors ont payé, vous pouvez dépenser l'argent.

Créez un budget qui détaille vos postes de dépenses et quand vous aurez besoin d'y faire appel. Je conseille d'obtenir trois devis pour les produits et services qui ne vous sont pas familiers, simplement afin de vous faire une idée sur ce qu'est un prix correct. Faites comprendre aux fournisseurs que vous contactez que vous faites jouer la concurrence.

Une fois que j'ai établi une relation de confiance avec une entreprise, j'ai tendance à faire des affaires avec eux d'une année sur l'autre. J'aime avoir de bonnes relations avec les fournisseurs

et je trouve que même si je paie un peu plus que si je faisais jouer la concurrence chaque année, je finis par gagner du temps et par obtenir un meilleur service de la part d'un vendeur qui me connaît bien.

Pour les petits évènements, vous pouvez garder une trace de vos dépenses dans un tableur assez simple. Pour les projets plus grands, demander à un comptable ou utiliser des logiciels de comptabilité peut être utile. Voici une liste des alternatives libres à Quicken (à différents niveaux et avec différents aspects !) : http://bit.ly/9RRgu0.

Le plus important est de garder une trace de toutes vos dépenses et de ne pas dépenser de l'argent que vous n'avez pas ! Si vous travaillez avec une organisation à but non lucratif pour gérer le budget de l'événement, demandez-lui de l'aide et des conseils avant de vous lancer.

Remercier

Il existe de nombreuses manières de remercier les gens et les entreprises qui ont apporté leur soutien à votre manifestation. Encore plus important, suivez toutes les promesses que vous avez faites dans le dossier. Communiquez à chaque fois qu'un engagement est tenu !

Durant la manifestation, trouvez des moyens d'entrer en contact avec les sponsors, en désignant un bénévole pour les accueillir et en les accueillant vous-même.

Après la manifestation, assurez-vous de remercier individuellement chaque sponsor et chaque bénévole pour sa contribution. Une association avec laquelle je travaille envoie des remerciements écrits à chaque sponsor en début d'année.

D'une manière générale, la communication est le terreau fertile de la levée de fonds. Porter attention aux sponsors et construire des relations authentiques avec eux aide à trouver plus de spon-

sors et à vous constituer une réputation de bon organisateur de manifestations.

Leçons apprises

Après avoir créé et animé des dizaines de manifestations, les deux aspects les plus importants que j'en tire ont été de trouver des mentors et d'apprendre à bien communiquer.

Les mentors m'ont aidée à transformer des coups de gueule en essais littéraires, du fouillis en dossiers et des conversations difficiles en perspectives. J'ai trouvé des mentors dans des entreprises qui parrainaient mes conférences, et me faisaient des retours détaillés, parfois pénibles. Et j'ai trouvé des mentors parmi les bénévoles qui passaient des centaines d'heures à écrire du logiciel pour mes manifestations, à recruter des orateurs, à documenter ce que nous étions en train de faire et à poursuivre la conférence après moi.

Apprendre à bien communiquer prend du temps, et c'est l'occasion de faire de nombreuses erreurs. J'ai appris à mes dépens que ne pas développer de relations avec les meilleurs sponsors signifie ne pas être financé l'année suivante ! J'ai aussi appris que les gens sont capables d'une formidable indulgence envers les erreurs, dès lors que vous communiquez tôt et souvent.

Bonne chance dans votre recherche de fonds, et merci de me dire si ce qui précède vous a aidé.

Le monde professionnel

38. Le Logiciel Libre dans l'administration publique

Till Adam

Issu du milieu de la musique et des sciences humaines, Till Adam a passé pratiquement les dix dernières années dans le monde de la programmation. Il travaille au sein de KDAB[1] *où il dirige plusieurs services, dont celui qui chargé des logiciels libres. Till officie aussi au sein du conseil d'administration de Kolab Systems AG, une entreprise dont le modèle économique repose entièrement sur les logiciels libres. Il vit avec sa femme et sa fille à Berlin.*

Introduction

Comme de nombreux autres auteurs de cette compilation d'articles, j'imagine, j'ai commencé à contribuer au logiciel libre lorsque j'étais étudiant. J'avais décidé relativement tard dans ma vie de poursuivre un cursus en informatique (ayant échoué à devenir riche et célèbre en tant que musicien). Je m'attendais donc à être légèrement plus âgé que mes pairs en obtenant mon diplôme. J'ai pensé qu'il serait bénéfique d'apprendre par moi-même la programmation, qui ne m'était pas trop enseignée à l'école, afin d'avoir plus d'atouts aux yeux de futurs employeurs, en dépit de mon âge. Après quelques incursions dans diverses

1 http://www.kdab.com/en-francais.

petites communautés, j’ai finalement trouvé ma voie dans le projet KDE et j’ai commencé à travailler sur l’application de courriel.

Grâce aux personnes extrêmement serviables et douées techniquement que j’y ai rencontrées, j’ai pu apprendre rapidement et contribuer de façon significative au code, ce qui m’a entraîné de plus en plus dans leur réseau social, mais aussi vers le domaine fascinant des problèmes techniques liés à la gestion de données personnelles.

Lorsque KDAB, une entreprise dont beaucoup d’employés utilisaient KDE, m’a demandé si, dans le cadre d’un stage étudiant, je voulais apporter mon aide sur la partie commerciale d’un projet en cours, j’ai bien sûr été ravi de pouvoir gagner ma vie et bidouiller le logiciel KDE en même temps. Au fil des ans, j’ai été témoin de l’adoption et de l’utilisation des architectures de gestion des données personnelles de KDE par le secteur public, particulièrement en Allemagne, où j’ai pu assister personnellement à la croissance économique de KDAB dans ce secteur géographique. Alors que j’évoluais vers des postes plus orientés vers le management, mon travail a finalement consisté notamment à vendre et livrer des services issus du logiciel libre, dont des produits KDE, à de grandes organisations, en particulier dans le secteur public.

Il faut noter que la majeure partie du travail sur le projet qui a inspiré ce texte était généralement fait en collaboration avec d’autres entreprises du logiciel libre, à savoir glOcode, un spécialiste de la cryptographie qui se charge du maintien de GNUPG, et Intevation, une entreprise de conseil qui se concentre exclusivement sur le logiciel libre ainsi que ses défis stratégiques et opportunités. Mention spéciale à Bernhard Reiter, l’un des fondateurs d’Intevation, qui a joué un rôle clé lors de la vente et de la conduite de bon nombre de ces projets. Les quelques fragments de sagesse contenus dans ce texte sont probablement issus de son

analyse et des nombreuses conversations que j'ai pu avoir avec lui au fil des ans.

Donc, si Bernhard et moi pouvions revenir dans le temps, quelles pourraient donc bien être les idées que nous partagerions avec nos « nous » plus jeunes et plus naïfs ? Eh bien, il s'avère qu'elles commencent toutes par la lettre « P ».

Personnes

Dans l'état actuel des choses, pour ceux qui travaillent dans les technologies de l'information et pour les décideurs, il est encore difficile d'utiliser du logiciel libre plutôt que des solutions propriétaires. Même en Allemagne, où le logiciel libre a un soutien politique relativement fort, il est plus facile et plus sûr de suggérer l'utilisation de quelque chose qui est perçu comme un « standard de l'industrie » ou comme « ce que tous les autres font » ; en d'autres termes, des solutions propriétaires.

Celui qui propose une solution en logiciel libre devra probablement affronter l'opposition de collègues moins aventureux (ou ayant moins de vision), l'examen minutieux des supérieurs, de plus grandes attentes par rapport aux résultats et une pression budgétaire irréaliste. Il faut donc un type particulier de personnes souhaitant prendre des risques personnels, potentiellement compromettre l'avancée de leur carrière et se lancer dans une bataille presque perdue d'avance. Ceci est bien sûr vrai dans n'importe quelle organisation. Mais, dans une administration publique, une ténacité particulière est requise car les choses bougent généralement plus lentement. Et une hiérarchie organisationnelle inflexible ajoutée à des options de carrière limitées amplifie le problème.

il s'avère quasiment impossible de faire prendre en compte les logiciels libres comme un choix possible sans alliés à l'intérieur. Si de telles personnes existent, il est important de les soutenir autant que possible dans leurs combats en interne. Ceci signifie

leur fournir des informations opportunes, fiables et vérifiables sur ce qui se passe dans la communauté avec laquelle l'organisation entend interagir. Ces informations doivent contenir suffisamment de détails pour fournir une image complète tout en atténuant la complexité de la communication et du chaos de planification qui caractérise parfois le travail dans le logiciel libre, de façon à ce que ça devienne plus gérable et moins effrayant. L'honnêteté et le sérieux aident à construire des relations fortes avec ces personnes-clés, qui sont la base du succès à plus long terme. Elles prennent appui sur vous, qui êtes leur intermédiaire avec le monde merveilleux et quelque peu effrayant des communautés du logiciel libre, pour trouver des chemins qui les mèneront elles et leurs organisations à leurs objectifs. Elles prennent également des décisions largement fondées sur la confiance personnelle. Cette confiance doit être acquise et conservée.

Afin d'y parvenir, il est important de ne pas se concentrer uniquement sur les résultats techniques des projets, mais de garder en tête les objectifs plus larges, personnels et organisationnels, que l'on doit atteindre lorsqu'on travaille sur ces projets. S'il fallait évaluer le succès ou l'échec du projet en cours, la capacité du chef de projet à se faire mousser de temps à autres devant ses supérieurs avec des fonctionnalités qui n'ont que peu d'importance dans le projet lui-même compte moins que le fait qu'un autre projet aura lieu ou non par la suite. Lorsque vous avez peu d'amis, les aider à réussir est un bon investissement.

Priorités

En tant que technophiles, les gens du logiciel libre ont tendance à se concentrer sur ce qui est nouveau, excitant et qui paraît important au niveau technologique. En conséquence de quoi, nous mettons moins l'accent sur les choses qui sont plus importantes dans le contexte d'une administration publique (souvent vaste). Mais considérez quelqu'un désireux de changer tout un ensemble de technologies dans une structure qui a plutôt tendance à rester

sur les mêmes technologies pendant une longue durée. Étant donné qu’un changement brusque est difficile et coûteux, il est de loin bien plus important d’avoir de la documentation sur les choses qui ne fonctionneront pas, de façon à pouvoir les éviter ou les contourner, que de savoir qu’une version à venir fonctionnera beaucoup mieux. Il est peu probable que cette nouvelle version soit jamais disponible pour les utilisateurs dont nous parlons ici. Et il est bien plus simple d’avoir affaire à des problèmes connus et anticipés plutôt que d’être forcé de faire face à des surprises. Le bogue documenté d’aujourd’hui est paradoxalement préférable à sa résolution de demain avec ses effets de bord imprévisibles.

Dans une grande organisation qui utilise des logiciels pendant une longue durée, le coût d’acquisition du logiciel, que ce soit par le biais de licences ou dans le cadre de développement sur mesure de logiciels libres par contrat, a peu d’importance en comparaison du coût de maintenance et de support. Cela mène à penser qu’au lieu de séduisantes nouveautés, complexes et sans doute moins matures, il vaut mieux proposer moins de fonctionnalités, plus stables, auxquelles on peut faire davantage confiance et qui ont moins besoin de maintenance intensive, ce qui entraîne une moindre charge pour l’organisme de support.

Alors que ces deux observations vont à l’encontre des instincts des développeurs de logiciels libres, ce sont ces mêmes aspects qui rendent attractif pour le secteur public le fait de payer pour le développement de logiciels libres, plutôt que de dépenser de l’argent pour des licences de produits pris au hasard. En partant d’une large palette de logiciels gratuitement disponibles, l’organisation peut investir son budget dans le perfectionnement des parties précises qui sont pertinentes pour ses propres opérations. Elle n’a ainsi pas à payer (via les coûts de licences) pour le développement de fonctionnalités clinquantes et guidées par le marché dont elle n’a pas besoin. En soumettant tout ce travail à la communauté en retour, la maintenance à long terme de ces améliorations et du logiciel de base est partagée par un grand

nombre de personnes. De plus, grâce au fait que ces améliorations deviennent publiques, d'autres organisations aux besoins similaires peuvent bénéficier de celles-ci sans coût supplémentaire. Cela maximise donc l'utilité de l'argent du contribuable, ce que toute administration publique souhaite (ou devrait souhaiter).

Politique d'acquisition

Si les budgets informatiques des agences gouvernementales sont clairement mieux utilisés dans l'amélioration du logiciel libre et dans son adaptation à leurs besoins, pourquoi est-ce si rarement ce que l'on fait ? L'équivalence des fonctionnalités pour les types de logiciels les plus utilisés a depuis longtemps été atteinte, la convivialité est la même, la robustesse et le coût total d'acquisition aussi. La notoriété et la connaissance sont bien sûr toujours des problèmes, mais le véritable obstacle à l'adoption de services en logiciel libre réside dans les conditions légales et administratives sous lesquelles cela doit se produire. Changer ces conditions nécessite du travail, au niveau de la politique et du lobbying. C'est rarement possible dans le contexte d'un projet individuel. Heureusement, des organisations telles que la Free Software Foundation Europe et sa sœur aux États-Unis font du lobbying en notre nom et font lentement changer les choses. Jetons un coup d'œil à deux problèmes centraux d'ordre structurel.

Des licences, pas des services

Beaucoup de budgets informatiques sont structurés de telle façon qu'une partie de l'argent est mise de côté pour l'achat d'un nouveau logiciel ou pour le paiement continu de l'utilisation d'un logiciel sous forme de licences. Comme il était inimaginable pour ceux qui ont construit ces budgets qu'un logiciel puisse être autre chose qu'un bien achetable, représenté par une licence propriétaire, il est souvent difficile ou impossible pour les décideurs informatiques de dépenser cette même somme d'argent pour des

services. La comptabilité de gestion n'en entendra simplement pas parler. Cela peut mener à la situation malheureuse où une organisation a la volonté et l'argent pour améliorer un logiciel libre afin qu'il convienne parfaitement à ses besoins, pour le déployer et pour le faire tourner pendant des années et envoyer ses contributions à la communauté, en retour, mais où cela ne peut se faire tant que toute l'affaire n'est pas enveloppée dans une vente et un achat artificiels et non nécessaires d'un produit imaginaire basé sur une licence libre.

Pièges légaux

Les cadres légaux pour les fournisseurs de logiciels supposent souvent que quiconque signant la production d'un logiciel exerce le plein contrôle des copyrights, marques déposées et brevets afférents. L'organisation cliente attend une garantie contre des risques variés de la part du fournisseur. Dans le cas où une société ou une personne produit une solution ou un service basé sur du logiciel libre, cela est souvent impossible car il y a d'autres titulaires de droits qui ne peuvent pas être raisonnablement impliqués dans l'arrangement contractuel. Ce problème apparaît plus ostensiblement dans le contexte des brevets logiciels. Il est pratiquement impossible pour un fournisseur de services de s'assurer contre les risques de contentieux de brevets, ce qui rend très risqué pour lui d'endosser la pleine responsabilité.

Prix

Historiquement, l'argument le plus décisif en faveur du logiciel libre donné au grand public a été l'économie financière qu'il permet. Le logiciel libre a en effet permis des économies à grande échelle pour beaucoup d'organisations depuis de nombreuses années. Le système d'exploitation GNU/Linux a été le fer de lance de ce développement. Ceci en raison de sa libre disponibilité au téléchargement qui a été perçue en opposition frappante

avec les licences onéreuses de son principal concurrent, Microsoft Windows.

Pour quelque chose d'aussi utilisé et utile qu'un système d'exploitation, il est indéniable que le bénéfice des coûts structurels vient des coûts de développement qui sont répartis sur de nombreuses parties. Malheureusement, l'espoir que ceci reste vrai pour tous les logiciels libres a mené à la pensée irréaliste que les coûts seront toujours réduits, largement et immédiatement. D'après notre expérience, ce n'est pas vrai. Comme nous l'avons vu dans les sections précédentes de cet ouvrage, il est très logique de tirer le meilleur parti de l'argent dépensé dans l'utilisation de logiciels libres et il est probable qu'au fil du temps et pour de nombreuses organisations de l'argent puisse être économisé. Mais pour une agence isolée qui cherche seulement à déployer un logiciel libre, il devra y avoir un investissement initial et un coût nécessaire associé pour obtenir le niveau de maturité et de robustesse nécessaire.

Alors que cela semble largement raisonnable aux professionnels de l'industrie informatique, il est souvent plus difficile de convaincre de cette vérité leurs supérieurs qui tiennent les cordons de la bourse. Surtout lorsque la potentielle économie financière a initialement été utilisée comme un argument pour faire entrer le logiciel libre, il peut s'avérer très difficile de gérer efficacement les attentes futures. Plus vite les décideurs sauront exactement de façon claire combien et dans quoi ils investissent, mieux ils accepteront de le faire sur le long terme.

Le meilleur rapport qualité/prix est toujours attirant et un fournisseur de services informatiques qui cessera d'être disponible parce que la forte pression sur les prix ne donne pas la réussite économique suffisante est aussi peu attractif dans le logiciel libre que dans les modèles économiques basés sur des licences propriétaires. Il est donc aussi dans l'intérêt des clients que les estimations de coûts soient réalistes et que les conditions économiques dans lesquelles le travail est effectué soient durables.

Conclusion

Notre expérience montre qu'il est possible de convaincre des organismes du secteur public de dépenser de l'argent dans des services basés sur des logiciels libres. C'est une proposition intéressante qui offre une plus-value et qui a un sens politique. Malheureusement, il existe encore des barrières structurelles. Mais avec l'aide de pionniers dans le secteur public, elles peuvent être contournées. Avec un soutien suffisant de notre part à tous, ceux qui travaillent pour le logiciel libre au niveau politique finiront par les surmonter. Une communication claire et honnête sur les réalités économiques et techniques peut favoriser des partenariats efficaces qui amènent des bénéfices à la communauté du logiciel libre, aux administrations publiques utilisant ces logiciels et à ceux qui les fournissent avec les services nécessaires dans une perspective viable et durable.

39. Trouver un modèle économique

Frank Karlitschek

*Frank Karlitschek est né en 1973 à Reutlingen, en Allemagne. Il a commencé à écrire des logiciels à l'âge de 11 ans. Il a étudié l'informatique à l'*université de Tübingen *et s'est impliqué dans le logiciel libre et les technologies de l'internet dans le milieu des années 1990. En 2001, il a commencé à contribuer à KDE en lançant* KDE-Look.org[1]*, un site communautaire d'œuvres qui deviendrait plus tard le réseau* openDesktop.org[2]. *Frank a initié plusieurs projets et initiatives* open source *comme* Social Desktop, Open Collaboration Services, Open-PC *et* ownCloud[3]. *En 2007, il a fondé une société appelée* hive01[4] *qui offrait des services et des produits autour de l'*open source *et des technologies de l'internet. Aujourd'hui, Frank est membre du conseil et vice-président de* KDE e.V.[5] *et c'est un intervenant habitué des conférences internationales.*

Introduction

Il y a dix ans, j'ai sous-estimé la valeur d'un modèle économique. Logiciel libre et modèle économique ? Deux concepts

1 http://kde-look.org.
2 http://opendesktop.org.
3 http://socialdesktop.org ; http://www.open-collaboration-services.org ; http://open-pc.com ; http://owncloud.org .
4 http://hive01.com.
5 http://ev.kde.org.

incompatibles. Du moins, c'est ce que je pensais lorsque j'ai commencé à contribuer à KDE en 2001. Le logiciel libre, c'est pour le plaisir et pas pour l'argent, n'est-ce pas ? Les libristes veulent un monde où chacun peut écrire du logiciel et où les grandes entreprises, telles que Microsoft ou Google, sont superflues. Tout logiciel devrait être libre et tous ceux qui souhaitent développer du logiciel devraient en avoir la possibilité – même les développeurs du dimanche. Donc, gagner de l'argent importe peu, n'est-ce pas ? Aujourd'hui, j'ai une opinion différente. Les développeurs devraient parfois être rémunérés pour leurs efforts.

Les raisons d'être du logiciel libre

La plupart des développeurs de logiciels libres ont deux principales motivations pour travailler sur le logiciel libre. La première est le facteur plaisir. C'est une expérience fantastique de travailler avec d'autres personnes très talentueuses du monde entier et de créer des technologies exceptionnelles. KDE, par exemple, est une des communautés les plus accueillantes que je connaisse. C'est tellement amusant de travailler avec des milliers de contributeurs du monde entier pour créer des logiciels qui seront utilisés par des millions de personnes. Pour faire simple, tout le monde est expert dans un ou plusieurs domaines et nous collaborons pour créer une vision partagée. Pour moi, c'est toujours génial de rencontrer d'autres contributeurs de KDE, d'échanger des idées ou de travailler ensemble sur nos logiciels, que nous nous rencontrions en ligne ou dans la vie réelle lors d'une des nombreuses conférences ou autres évènements. Et il s'agit aussi d'amitié. Au fil des années, je me suis fait beaucoup de bons amis au sein de KDE.

Mais les contributeurs de KDE ne sont pas uniquement motivés par le plaisir de rejoindre KDE. Il y a aussi la conviction que chacun de nous peut rendre le monde meilleur par ses contributions. Le logiciel libre est essentiel si vous vous souciez de l'accès à la technologie et à l'informatique pour les pays en voie

de développement. Cela permet aux personnes pauvres d'avoir leur place dans l'ère de l'information sans acheter des licences coûteuses pour des logiciels propriétaires. Il est essentiel pour les personnes qui se soucient de la confidentialité et de la sécurité, parce que le logiciel libre est le seul et unique moyen de savoir exactement ce que votre ordinateur fait avec vos données privées. Le logiciel libre est important pour un écosystème informatique sain, parce qu'il permet à tout le monde de bâtir à partir du travail des autres et de vraiment innover. Sans le logiciel libre, il n'aurait pas été possible à Google ou Facebook de lancer leurs entreprises. Il n'est pas possible d'innover ni de créer la prochaine technologie révolutionnaire si vous dépendez de logiciels propriétaires et que vous n'avez pas accès à toutes les parties du logiciel.

La nécessité d'un écosystème

Voilà les principales raisons pour lesquelles je veux que le logiciel libre et particulièrement le bureau libre soient largement répandus. Pour y parvenir, il nous faut bien plus de contributeurs qu'aujourd'hui. Par contributeurs, j'entends des gens qui écrivent les infrastructures centrales, le bureau et les applications majeures. Nous avons besoin de gens qui travaillent sur l'utilisabilité, sur les illustrations, sur la promotion et sur bien d'autres aspects importants. KDE est déjà une grande communauté avec des milliers de membres. Mais nous avons besoin de davantage de gens pour aider à rivaliser de manière sérieuse avec le logiciel propriétaire.

La communauté du logiciel libre est minuscule comparée au monde du logiciel propriétaire. D'un côté, ce n'est pas un problème car le modèle de développement logiciel distribué du monde du logiciel libre est bien plus performant que la façon d'écrire du logiciel à sources fermées. Un grand avantage est, par exemple, la possibilité de mieux réutiliser du code. Mais même avec ces avantages, nous avons besoin de bien plus de contribu-

teurs qu'aujourd'hui si nous voulons réellement conquérir le marché de l'ordinateur de bureau et celui du mobile.

Nous avons aussi besoin d'entreprises pour nous aider à apporter notre travail sur le marché de masse. Bref, nous avons besoin d'un vaste écosystème en pleine forme qui permette de vivre en travaillant sur le logiciel libre.

La situation actuelle

J'ai commencé à contribuer à KDE il y a plus de 10 ans et, depuis, j'ai vu d'innombrables volontaires très motivés et talentueux rejoindre KDE. C'est vraiment génial. Le problème, c'est que j'ai aussi vu beaucoup de contributeurs expérimentés abandonner KDE. C'est vraiment triste. Parfois, c'est simplement la marche normale du monde : les priorités changent et les gens se concentrent sur autre chose. Le problème, c'est que beaucoup abandonnent aussi à cause de l'argent. Il arrive un moment où les gens décrochent leur diplôme et veulent avoir autre chose que leur chambre d'étudiant.

Plus tard, ils veulent se marier et avoir des enfants. À partir de là, ils doivent trouver du travail. Il y a quelques entreprises dans l'écosystème de KDE qui proposent des postes liés à KDE. Mais cela ne représente qu'une petite part des emplois disponibles dans le secteur informatique. Du coup, beaucoup de membres chevronnés de KDE doivent travailler dans des entreprises où ils doivent utiliser des logiciels propriétaires qui n'ont rien à voir avec KDE ou le logiciel libre. Tôt ou tard, la plupart de ces développeurs abandonnent KDE. J'ai sous-estimé cette tendance il y a 10 ans, mais je pense que c'est un problème pour KDE sur le long terme, parce que nous perdons nos membres les plus expérimentés au profit des entreprises de logiciel propriétaire.

Le monde de mes rêves

Dans le monde idéal que j'imagine, les gens peuvent payer leur loyer en travaillant sur les logiciels libres et ils peuvent le faire de telle sorte que ça n'entre pas en conflit avec nos valeurs. Ceux qui contribuent à KDE devraient avoir tout le temps qu'ils veulent pour contribuer à KDE et au monde libre en général. Ils devraient gagner de l'argent en aidant KDE. Leur passe-temps deviendrait leur travail. Cela permettrait à KDE de se développer de manière spectaculaire, parce que ce serait super de contribuer et d'avoir en même temps de bonnes perspectives d'emploi stable et à long terme.

Quelles possibilités avons-nous ?

Du coup, quelles sont les solutions possibles ? Que pouvons-nous faire pour les mettre en œuvre ? Y a-t-il des moyens pour que les développeurs paient leur loyer tout en travaillant sur du logiciel libre ? Je voudrais exposer ici quelques idées que j'ai rassemblées au cours de plusieurs discussions avec des contributeurs au logiciel libre. Certaines d'entre elles sont probablement polémiques, parce qu'elles introduisent des idées complètement neuves au sein du monde du logiciel libre. Mais je pense qu'il est essentiel pour nous de voir au-delà de notre monde actuel si nous voulons mener à bien notre mission.

Du développement sponsorisé

Aujourd'hui, de plus en plus d'entreprises apprécient l'importance du logiciel libre et contribuent à des projets de logiciels libres, ou lancent même leurs propres projets de logiciel libre. C'est une chance pour les développeurs de logiciels libres. Nous devrions parler à davantage d'entreprises et les convaincre de s'associer au monde du logiciel libre.

Des dons de la part des utilisateurs

Il devrait y avoir une manière facile pour les utilisateurs de donner de l'argent directement aux développeurs. Si un utilisateur d'une application populaire veut soutenir le développeur et promouvoir ses développements à venir pour cette application, donner de l'argent devrait ne tenir qu'à un clic de souris. Le système de dons peut être construit au sein même de l'application pour rendre le don d'argent aussi facile que possible.

Des primes

L'idée derrière les primes est qu'un ou plusieurs utilisateurs d'une application peuvent payer pour le développement d'une fonctionnalité particulière. Un utilisateur peut soumettre la liste de ses demandes de nouvelles fonctionnalités sur un site web et annoncer combien il est prêt à payer pour cela. D'autres utilisateurs qui apprécient ces propositions pourraient ajouter de l'argent à la demande de fonctionnalité. Au bout d'un moment, le développeur commence à mettre au point la fonctionnalité et récupère l'argent des utilisateurs. Cette possibilité de primes n'est pas facile à introduire dans le processus. Des gens ont déjà essayé de mettre en place quelque chose de similaire, sans succès. Mais je pense que ça peut marcher si on s'y prend bien.

Du support

L'idée est que le développeur d'une application vende directement du support aux utilisateurs de l'application. Par exemple, les utilisateurs d'une application achètent du support pour, disons, 5 € par mois et obtiennent le droit d'appeler directement le développeur à des plages horaires spécifiques de la journée, ils peuvent poser des questions à une adresse de courriel spécifique, ou le développeur peut même aider les utilisateurs par le biais d'un bureau à distance. J'ai bien conscience que beaucoup de développeurs n'aimeront pas l'idée que les utilisateurs puissent les appeler et leur poser des questions bizarres. Mais si cela signi-

fie qu'ils gagnent suffisamment avec le système de support pour travailler à plein temps sur leurs applications, alors c'est certainement une bonne chose.

Des soutiens

L'idée c'est que les utilisateurs finaux puissent devenir les soutiens d'une application. Le bouton « Soutenez ce projet » pourrait être intégré directement dans l'application. L'utilisateur devient alors un soutien par un paiement mensuel de, par exemple, 5 € qui vont directement au développeur. Tous les soutiens sont listés dans la fenêtre « À propos de l'application » avec leurs photos et leurs noms réels. Une fois par an, tous les soutiens sont aussi invités à une fête spéciale avec les développeurs. Il est possible qu'un développeur puisse devenir capable de travailler à plein temps sur une application, si assez d'utilisateurs deviennent des soutiens.

Des programmes de fidélité

Certaines applications ont intégré des services web, et certains de ces services web exécutent des programmes affiliés. Par exemple, un lecteur multimédia peut être intégré à la boutique en ligne de MP3 Amazon ou un lecteur PDF peut être intégré à une boutique en ligne de livres numériques. À chaque fois qu'un utilisateur achète du contenu via cette application, le développeur obtient un peu d'argent.

Des magasins d'applications sous forme de binaires

Beaucoup de gens ne savent pas qu'il est possible de vendre des binaires de logiciels libres. La licence GPL exige simplement de fournir également le code source. Il est donc parfaitement légal de vendre des binaires bien empaquetés de notre logiciel. En réalité, les sociétés comme Red Hat et Novell vendent déjà notre logiciel dans leurs distributions commerciales. Mais les développeurs n'en bénéficient pas directement. Tous les revenus vont aux

sociétés et rien ne va aux développeurs. On devrait donc permettre aux développeurs de logiciels libres de vendre à l'utilisateur final des applications bien empaquetées, optimisées et testées. Cela pourrait particulièrement bien fonctionner pour Mac ou Windows. Je suis sûr qu'un tas de gens seraient prêts à payer quelque chose pour des binaires d'Amarok pour Windows ou de digiKam pour Mac, si tout l'argent allait directement au développeur.

Conclusion

La plupart de ces idées ne sont pas faciles à mettre en œuvre. Cela nécessite de modifier notre logiciel, nos méthodes de travail et même nos utilisateurs, qu'il faut encourager à montrer qu'ils apprécient le logiciel que nous créons, en nous aidant à financer son développement.

Cependant, les bénéfices potentiels sont énormes. Si nous pouvons assurer des sources de revenus pour notre logiciel, nous pouvons conserver nos meilleurs contributeurs et peut-être en attirer de nouveaux. Nos utilisateurs auront un meilleur produit avec un développement logiciel plus rapide, ils pourront influencer directement le développement par le biais de primes et ils bénéficieront d'un meilleur support.

Le logiciel libre n'est plus seulement un loisir sur votre temps libre. Il est temps d'en faire un business.

40. La réalité économique du logiciel libre

Carlo Daffara

*Carlo Daffara est chercheur dans les domaines des modèles économiques basés sur l'*open source*, du développement collaboratif d'objets numériques et de l'utilisation de logiciels* open source *dans les entreprises. Il fait partie du comité éditorial de relecture du journal international des logiciels et processus* open source *(International Journal of Open Source Software & Processes : IJOSSP). Il est également membre du comité technique de deux centres régionaux de compétences* open source *et du réseau juridique européen FSFE (fondation européenne pour le logiciel libre). Il a pris part aux comités SC34 et JTC1 pour la branche italienne de l'ISO, UNINFO, et a travaillé au sein du groupe de travail de la société Internet du logiciel public (Internet Society Public Software) ainsi que pour beaucoup d'autres initiatives liées à la normalisation. Auparavant, Carlo Daffara était le représentant italien dans le groupe de travail européen sur le logiciel libre, la première initiative de l'Union européenne afin de soutenir l'*open source *et le logiciel libre. Il a présidé le groupe de travail SME du groupe d'étude de l'UE sur la compétitivité et le groupe de travail IEEE des intergiciels open source du comité technique sur le calcul évolutif. Il a travaillé en tant qu'examinateur du projet pour la Commission européenne dans le domaine de la coopération internationale, l'ingénierie logicielle, l'open source et les systèmes distribués et a été directeur de recherche dans plusieurs projets de recherche de l'Union européenne.*

Introduction

« Comment gagner de l’argent avec le logiciel libre ? » était une question très courante, il y a encore seulement quelques années. Désormais, cette question s’est transformée en « quelles sont les stratégies commerciales pouvant être mises en œuvre en se basant sur le logiciel libre et *open source* ? ». Cette question n’est pas aussi anodine qu’elle y paraît, puisque de nombreux chercheurs universitaires écrivent encore ce genre de textes : « le logiciel *open source* est délibérément développé hors de tout mécanisme de marché… il échoue à contribuer à la création de valeur aux développements, contrairement au marché du logiciel commercial… il ne génère pas de profit, de revenus, d’emplois ou de taxes…. Les licences *open source* sur les logiciels visent à supprimer les droits d’auteur sur le logiciel et empêchent d’établir un prix pour le logiciel. Au final, les logiciels développés ne peuvent être utilisés pour générer des profits. » [Koot 03] ou [Eng 10] indiquent que « des économistes ont montré que les collaborations *open source* dans le monde réel s’appuient sur plusieurs incitations différentes telles qu’enseigner, se démarquer et se créer une réputation » (sans parler des incitations économiques). Cette vue purement « sociale » du logiciel libre et *open source* est partiale et fausse. Et nous démontrerons qu’il y a des raisons économiques liées au succès des métiers du libre et de l’*open source* qui vont au-delà des collaborations purement bénévoles.

Le logiciel libre et *open source* et les réalités économiques

Dans la plupart des domaines, l’utilisation d’un logiciel libre et *open source* apporte un avantage économique substantiel, grâce au développement partagé et aux coûts de maintenance, déjà décrits par des chercheurs comme Gosh, qui a estimé une réduction de coût de 36 % en R&D[1]. La vaste part de marché des déploiements « internes » de logiciels libres et *open source*

1 [NdT] « Recherche et Développement ».

explique pourquoi certains des bénéfices économiques ne sont pas directement visibles sur le marché des services commerciaux.

L'étude FLOSSIMPACT a montré, en 2006, que les entreprises qui contribuent au code de projets de logiciels libres et *open source* ont, au total, au moins 570 000 employés et un chiffre d'affaires annuel de 263 milliards d'euros [Gosh 06], faisant ainsi du logiciel libre et *open source* l'un des phénomènes les plus importants des NTIC. Il est important aussi de reconnaître qu'un pourcentage non négligeable de cette valeur économique n'est pas directement perceptible du marché, vu que la majorité du logiciel n'est pas développée dans l'intention de le vendre (le soi-disant logiciel « prêt à l'emploi ») mais uniquement à usage interne. Comme le réseau thématique FISTERA EU l'a identifié, la majorité du logiciel est en réalité développée seulement pour un usage interne.

Il est clair que ce qu'on appelle « le marché logiciel » est en réalité bien plus réduit que le vrai marché du logiciel et des services et que 80 % restent invisibles. Nous verrons que le FLOSS tient une place économique importante de ce marché, directement grâce à ce modèle de développement interne.

Région	**Licences de logiciels propriétaires**	**Services logiciel (développement personnalisation)**	**Développement interne**
Union européenne	19 %	52 %	29 %
États-Unis	16 %	41 %	43 %
Japon	N/A	N/A	32 %

Modèles économiques et proposition de valorisation

L'idée de base d'un modèle économique est assez simple : j'ai quelque chose ou je peux faire quelque chose (la « proposition de valeur ») et c'est plus rentable pour vous de me payer ou d'obtenir ce quelque chose plutôt que de le faire vous-même (il est même parfois impossible de trouver des alternatives, comme dans le cas de monopoles naturels ou créés par l'homme, et l'idée même de le produire par soi-même n'est pas envisageable). Il y a deux sources possibles de valeur : une propriété (quelque chose qui peut être échangé) et l'efficacité (quelque chose propre à ce que fait une entreprise et la manière dont elle le fait).

Avec l'*open source*, la « propriété » est généralement non exclusive (à l'exception de ce qui est nommé « le noyau ouvert », où une partie du code n'est pas libre du tout et cela sera abordé plus loin dans cet article). D'autres exemples de propriété concernent le droit des marques, les brevets, les licences… tout ce qui peut être transféré à une autre entité par contrat ou par une transaction légale. L'efficacité est la capacité à effectuer une action avec un coût moindre (qu'il soit tangible ou intangible) et cela correspond à la spécialisation dans un domaine d'application ou apparaît grâce à une nouvelle technologie.

Pour le premier cas, un simple exemple est celui de la réduction du temps nécessaire à la réalisation d'une action quand vous augmentez votre expertise sur un sujet donné. La première fois que vous installez un système complexe, cela peut demander beaucoup d'efforts et cet effort diminue d'autant plus que vous connaissez les tâches nécessaires pour réaliser l'installation elle-même. Pour le second, cela peut être l'apparition d'outils qui simplifient le processus (par exemple, avec le clonage d'images) et introduisent une importante rupture, un « saut » dans la courbe efficacité-temps.

Ces deux aspects sont la base de tous les modèles économiques que nous avons analysés par le passé ; il est possible de montrer

que tous s’inscrivent dans une logique de continuité entre les propriétés et l’efficacité.

Parmi les résultats de notre précédent projet de recherche, nous avons trouvé que les projets basés sur un modèle propriétaire ont tendance à obtenir moins de contributions extérieures car une opération juridique est nécessaire pour faire partie des propriétés de l’entreprise. Pensez par exemple aux licences doubles : afin que son code fasse partie du code du produit, un contributeur extérieur doit renoncer à ses droits sur son code afin que l’entreprise puisse vendre la version commerciale autant que la version *open source*.

D’un autre côté, les modèles totalement orientés sur l’efficacité ont tendance à avoir plus de contributions et de visibilité mais des résultats financiers plus faibles. Je l’ai écrit plusieurs fois : il n’y a pas de modèle économique idéal mais un éventail de modèles possibles et les entreprises devraient s’adapter elles-mêmes pour changer les conditions du marché et aussi adapter leur modèle. Certaines entreprises débutent par des modèles entièrement axés sur l’efficacité puis construisent, avec le temps, une propriété en interne, d’autres ont commencé avec un modèle orienté vers la propriété et ont évolué différemment pour augmenter les contributions et réduire les efforts d’ingénierie (ou développer la base d’utilisateurs afin de créer d’autres moyens d’avoir un retour financier grâce aux utilisateurs).

Une typologie des modèles économiques

L’étude EU FLOSSMETRICS des modèles économiques basés sur le logiciel libre a identifié, après analyse de plus de 200 entreprises, une taxonomie des principaux modèles économiques utilisés par les entreprises *open source*. Les principaux modèles identifiés sur le marché sont :

La double licence : le même code source est distribué sous GPL et sous une licence propriétaire. Ce modèle est principale-

ment utilisé par les producteurs de logiciels et outils pour développeurs et fonctionne grâce à la robuste clause de couplage de la GPL, qui requiert que les travaux dérivés et logiciels liés directement soient distribués sous la même licence. Les entreprises ne souhaitant pas distribuer leur propre logiciel sous GPL peuvent obtenir une licence propriétaire leur octroyant une exemption des conditions de la GPL, ce qui semble souhaitable à certains. L'inconvénient de cette licence double est que les contributeurs externes doivent accepter des conditions similaires et cela a révélé des réductions de contributions externes, se limitant à des corrections de bogues et des ajouts mineurs.

Le modèle « noyau ouvert » (précédemment appelé « valeur ajoutée propriétaire » ou « séparation entre libre et propriétaire ») : ce modèle fait la différence entre un logiciel libre basique et une version propriétaire qui repose sur la version libre mais avec l'ajout de greffons propriétaires. La plupart des entreprises qui suivent un tel modèle adoptent la *Mozilla Public License*, car elle permet explicitement cette forme de mélange et permet une plus grande participation des contributions externes sans les mêmes contraintes de consolidation du droit d'auteur que dans l'usage de doubles licences. Ce modèle a un inconvénient intrinsèque : le logiciel libre doit être de grande valeur pour être attractif pour les utilisateurs, autrement dit il ne doit pas être réduit à une version aux possibilités limitées. Pourtant, dans le même temps, il ne doit pas « cannibaliser » le produit propriétaire. Cet équilibre est difficile à atteindre et à maintenir dans la durée. En outre, si le logiciel est d'un grand intérêt, les développeurs peuvent essayer d'apporter les fonctionnalités manquantes dans le logiciel libre, réduisant ainsi l'intérêt de la version propriétaire et donnant potentiellement naissance à un logiciel concurrent entièrement libre qui ne souffrira pas des mêmes limitations.

Les experts produits : des entreprises qui ont créé ou maintiennent un projet logiciel spécifique et utilisent une licence libre

pour le distribuer. Les principaux revenus viennent du service, comme la formation ou l'expertise, et suivent la classification EUWG d'origine « le meilleur code vient d'ici » et « les meilleures compétences sont ici » [DB 00]. Cela conforte l'impression courante que les experts les plus compétents sur un logiciel sont ceux qui l'ont développé et qu'ils peuvent ainsi fournir des services au prix d'un minimum de commercialisation, s'appuyant sur la fourniture gratuite du code. L'inconvénient de ce modèle est que le coût d'entrée pour des concurrents potentiels est faible, vu que le seul investissement nécessaire est l'acquisition des compétences sur le logiciel lui-même.

Les fournisseurs de plateforme : des entreprises qui apportent un ensemble de services, avec support et intégration de certains projets, constituant une plateforme cohérente et testée. En ce sens, même les distributions GNU/Linux sont considérées comme de plateformes ; un point intéressant : ces distributions sont diffusées en grande partie sous licence libre pour maximiser les contributions externes. Elles s'appuient sur la protection du droit d'auteur pour empêcher la copie sauvage sans empêcher les « déclinaisons » (suppression des particularités soumises à droit d'auteur comme les logos ou droit des marques, pour créer un nouveau produit). Des exemples de clones de Red Hat sont CentOS et Oracle Linux. La valeur ajoutée provient d'une qualité garantie, de la stabilité et de la fiabilité ainsi que d'une garantie de support pour les applications métier critiques.

Les entreprises de conseil et de recrutement : les entreprises de cette catégorie ne font pas vraiment de développement mais fournissent des conseils de sélection et des services d'évaluation pour une vaste gamme de projets, d'une manière qui est proche du rôle de l'analyste. Ces entreprises ont tendance à avoir un impact très limité sur les communautés car les résultats de l'évaluation et du processus d'évaluation sont généralement des données propriétaires.

Les fournisseurs de support global : des entreprises qui proposent un support centralisé pour un ensemble de produits de logiciel libre, généralement en employant directement les développeurs ou en remontant les demandes de support.

La validation juridique et l'expertise : ces entreprises n'apportent pas de développement de code source mais fournissent une aide à la vérification de conformité aux licences, parfois en apportant une garantie et une assurance contre les attaques juridiques ; certaines entreprises utilisent des outils pour assurer que le code n'est pas réutilisé.

La formation et la documentation : des entreprises qui proposent de la formation, en ligne et en présentiel, des documentations et des manuels supplémentaires. Cela est généralement fourni dans le cadre d'un contrat de support. Mais récemment, quelques réseaux de centres de formation ont lancé des cours orientés spécifiquement vers le logiciel libre.

Le partage des coûts de R&D : une entreprise ou une société peut avoir besoin d'une nouvelle version ou d'une amélioration d'un paquet logiciel et financer un consultant ou un développeur pour réaliser le travail. Plus tard, le logiciel développé est redistribué en *open source* pour bénéficier de l'ensemble des développeurs expérimentés pouvant le déboguer et l'améliorer.

Un bon exemple est la plateforme Maemo[1], utilisée par Nokia pour ses smartphones (comme le N810) ; au sein de Maemo, seul 7,5 % du code est propriétaire, apportant une réduction des coûts estimée à 228 millions de dollars (et une réduction du temps de mise sur le marché d'un an).

Un autre exemple est l'écosystème Eclipse, un environnement de développement intégré (EDI) distribué à l'origine par IBM comme logiciel libre puis ensuite géré par la fondation Eclipse. De nombreuses entreprises ont choisi Eclipse comme infrastructure pour leur produit et ont ainsi réduit le coût global pour la

1 http://fr.wikipedia.org/wiki/Maemo.

création d'un logiciel fournissant une fonctionnalité pour les développeurs. Il y a un grand nombre d'entreprises, d'universités et de particuliers qui participent à l'écosystème Eclipse.

Comme récemment constaté, IBM contribue aux alentours de 46 % au projet, les contributeurs à titre personnel représentant 25 % et un grand nombre d'entreprises comme Oracle, Borland, Actuate et de nombreuses autres ayant des participations allant de 1 à 7 %. Ceci est semblable aux résultats obtenus grâce à l'analyse du noyau Linux, qui montrent que, lorsqu'il y a un écosystème sain et de grande taille, le partage des tâches réduit de manière significative les coûts de maintenance.

Dans [Gosh 06], on estime qu'il est possible de faire des économies de l'ordre de 36 % dans la recherche et la conception logicielle grâce à l'utilisation du logiciel libre. Ces économies constituent en elles-mêmes le plus gros « marché » réel pour le logiciel libre, ce qui est démontré par le fait qu'au moins une partie du code des développeurs est basé sur du logiciel libre (56,2 % comme mentionné dans [ED 05]). Un autre excellent exemple de « coopétition » inter-entreprises est le projet WebKit, le moteur de rendu HTML à la base du navigateur Google Chrome ainsi que d'Apple Safari et qui est utilisé dans la majorité des appareils mobiles. Dans ce projet, après un délai initial d'un an, le nombre de contributions externes a commencé à devenir significatif et, après un an et demi, il surpasse largement les contributions d'Apple – réduisant de fait les coûts de maintenance et d'ingénierie grâce à la répartition des tâches entre les co-développeurs.

Les revenus indirects : une entreprise peut choisir de financer des projets de logiciel libre si ces projets peuvent créer une source de revenus importante pour des produits dérivés, non liés directement au code source ou au logiciel. L'un des cas les plus courants correspond à l'écriture de logiciel nécessaire au fonctionnement de matériel, par exemple, les pilotes d'un système d'exploitation pour un matériel spécifique. En fait, de nombreux fabricants de

matériel distribuent déjà gratuitement leurs pilotes logiciels. Certains d'entre eux distribuent déjà certains de leurs pilotes (surtout ceux pour le noyau Linux) sous une licence libre.

Le modèle du produit d'appel est une stratégie commerciale traditionnelle, répandue même à l'extérieur du monde du logiciel : dans ce modèle, les efforts sont consacrés à un projet de logiciel libre et *open source* afin de créer ou d'étendre un autre marché dans des conditions différentes. Par exemple, les fournisseurs de composants matériels investissent dans le développement de pilotes logiciels pour des systèmes d'exploitation libres (comme GNU/Linux) pour s'étendre sur le marché spécifique des composants.

D'autres modèles de revenus auxiliaires sont ceux, par exemple, de la fondation Mozilla qui réunit une somme d'argent non négligeable grâce à un partenariat avec Google sur le moteur de recherche (pour un montant estimé à 72 millions de dollars en 2006), tandis que SourceForge/OSTG est financé en majorité par les recettes des ventes en ligne du site partenaire ThinkGeek.

Certaines entreprises ont plus d'un modèle principal et sont par conséquent comptées en double. C'est particulièrement le cas de la plupart des entreprises qui utilisent une licence double et vendent aussi du service de support. En outre, les experts d'un produit ne sont comptés que s'ils ont une partie visible de leur entreprise qui contribue au projet en tant que « *commiter* principal ». Autrement, le nombre d'experts serait bien plus élevé, du fait que certains projets sont au cœur du support commercial de nombreuses entreprises (de bons exemples sont OpenBravo et Zope).

Il faut aussi tenir compte du fait que les fournisseurs de plateforme, même s'ils sont limités en nombre, tendent à avoir des taux de facturation plus élevés que les experts ou que les entreprises à noyau ouvert. De nombreux chercheurs essaient de savoir s'il existe un modèle plus « efficace » parmi ceux qui sont

mentionnés. Selon nos analyses, le futur le plus probable sera l'évolution d'un modèle à l'autre, avec une consolidation sur le long terme des consortiums de développement (comme les fondations Eclipse et Apache) qui fournissent une forte infrastructure légale et des avantages de développement ainsi que des spécialistes apportant des offres verticales pour des marchés spécifiques.

Conclusion

Le logiciel libre et *open source* permet non seulement une présence pérenne, et même très large, sur le marché (Red Hat est déjà proche du milliard de dollars de revenus annuels), mais aussi plusieurs modèles différents qui sont totalement impossibles avec le logiciel propriétaire. Le fait que le logiciel libre et *open source* soit un bien « non-rival » facilite aussi la coopération entre entreprises, tant pour accroître sa présence mondiale que pour signer des contrats à grande échelle pouvant demander des compétences multiples, à la fois sur le plan géographique (même produit ou service, région géographique différente), et de façon « verticale » (entre produits) ou « horizontale » (des domaines d'application).

La facilité avec laquelle on peut créer de nouveaux écosystèmes est l'une des raisons qui expliquent pourquoi le logiciel libre et *open source* fait partie intégrante de la plupart des infrastructures informatiques dans le monde. Et comment il enrichit et aide les entreprises et administrations publiques à réduire leurs coûts et à collaborer pour de meilleurs logiciels.

Bibliographie

[DB00] DAFFARA, C. BARAHONA, J.B. *Free Software/Open Source: Information Society Opportunities for Europe working paper*, http://eu.conecta.it paper, OSSEMP workshop, Third international conference on open source. Limerick 2007
[ED05] EVANS Data, Open Source Vision report, 2005

[Eng10] ENGELHARDT S. MAURER S. *The New (Commercial) Open Source: Does it Really Improve Social Welfare* Goldman School of Public Policy Working Paper No.GSPP10-001, 2010

[Gar06] GARTNER GROUP, *Open source going mainstream. Gartner report*, 2006

[Gosh06] GOSH, et al. *Economic impact of FLOSS on innovation and competitiveness of the EU ICT sector*. http://bit.ly/cNwUz0

[Koot03] KOOTHS, S. LANGENFURTH, M. KAIWEY, N. *Open-Source Software: An Economic Assessment Technical report*, Muenster Institute for Computational Economics (MICE), University of Muenster

Aspects juridiques

41. Le rôle du juriste dans le logiciel libre et *open source*

Till Jaeger

Le docteur Till Jaeger est avocat associé du cabinet JBB Rechtsanwaelte depuis 2001. Sa spécialisation en droit d'auteur et droit des médias lui permet de conseiller aussi bien les grandes et moyennes entreprises du secteur des technologies de l'information que les institutions gouvernementales et les développeurs de logiciels dans des affaires qui impliquent contrats, licences et usage en ligne. Une des facettes particulières de son travail concerne les litiges judiciaires relatifs aux logiciels libres et open source. *Il est co-fondateur de l'institut pour l'étude juridique du logiciel libre et* open source *(ifrOSS). Il aide aussi développeurs et éditeurs de logiciels à mettre en compatibilité et en conformité leurs licences libres. Till a représenté le projet gpl-violations.org dans plusieurs actions en justice qui portaient sur le respect de la GPL. Il a également publié divers articles et livres à propos de questions juridiques sur les logiciels libres et* open source. *Il a été membre du comité C lors de l'élaboration de la GPLv3.*

Soyons clair d'emblée : je ne suis pas un geek, ne l'ai jamais été, et n'ai aucune intention de le devenir.

Je suis juriste. Il est probable que la plupart des lecteurs de cet ouvrage auront tendance à témoigner plus de sympathie envers les geeks qu'envers les juristes. Mais c'est un fait et je ne veux pas m'en cacher. Que la communauté du logiciel libre et *open source* n'ait pas d'intérêt particulier pour les juristes, car elle est trop

occupée à développer du code, j’en étais déjà conscient quand nos chemins se sont croisés pour la première fois au début de l’année 1999. Cependant, il y avait encore un certain nombre de choses que j’ignorais.

En 1999, alors que je terminais ma thèse de doctorat portant sur le droit d’auteur classique, j’évaluais la portée des droits moraux. Dans cette perspective, j’ai consacré un bon moment à examiner la question suivante : comment les droits moraux des développeurs sont-ils protégés par la licence GPL, qui donne aux utilisateurs le droit de modifier leurs logiciels ? C’est ainsi que je suis entré pour la première fois en contact avec le logiciel libre et *open source*.

À l’époque, les termes « libre » et « ouvert » avaient certainement des significations différentes. Mais cette distinction ne donnait pas matière à argumentation dans le monde où je vivais. Pourtant, comme j’étais libre de faire ce qui m’intéressait et ouvert à l’exploration de nouvelles questions sur le droit d’auteur, j’ai rapidement découvert que les deux termes ont bien quelque chose en commun : ils sont certes différents mais il est pourtant bien mieux de les utiliser ensemble…

Voici trois choses que j’aurais souhaité savoir à l’époque.

D’abord, que mes connaissances techniques, spécialement dans le domaine du logiciel, étaient insuffisantes. Ensuite, que je ne connaissais pas vraiment la communauté ni ce qui importait pour ses membres. Enfin et surtout, que je ne connaissais alors pas grand-chose aux juridictions étrangères. Il m’aurait été bien utile de savoir tout cela dès le départ.

Depuis lors, j’ai appris pas mal de choses et de même que la communauté partage bien volontiers ses réalisations, je suis heureux de partager à mon tour les leçons que j’ai tirées[1].

1 L’*Institut für Rechtsfragen der Freien und Open Source Software* (Institut des questions de droit sur les logiciels libres et *open source*) propose, entre autres, une collection d’ouvrages et de jurisprudences en lien avec les logiciels libres et *open source* ; pour plus de détails, voir sur le site http://www.ifross.org.

Les connaissances techniques

Comment est formée une architecture logicielle ? À quoi ressemble la structure technique d'un logiciel ? Quelles licences sont compatibles ou incompatibles entre elles, comment et pourquoi ? Comment est structuré le noyau Linux ?

Pour ne mentionner qu'un exemple, savoir ce qui constitue une « œuvre dérivée » selon la GPL est une question importante qui détermine la façon dont le logiciel peut être placé sous une licence. Tout élément rentrant dans le champ d'une œuvre dérivée d'un logiciel préexistant sous la GPL doit être redistribué selon les termes de cette licence. Juger si un programme donné est une « œuvre dérivée » ou non nécessite une compréhension technique approfondie. Ainsi, l'interaction des modules de programmes, des liaisons, des IPC (Communications inter-processus), des greffons, des infrastructures technologiques, des fichiers d'en-tête, etc. détermine (entre autres critères) le degré de connexité d'un programme par rapport à un autre, ce qui aide à déterminer s'il est ou non dérivé d'un autre programme.

La connaissance de l'industrie et de la communauté

Au-delà de ces problèmes de fonctionnalités, je n'avais aucune connaissance approfondie des principes et valeurs du libre et de l'*open source* ni de la motivation des développeurs et des entreprises qui utilisent du logiciel libre. Je ne connaissais guère mieux son arrière-plan philosophique, et n'étais pas plus au fait des questions pratiques telles que : « que fait un mainteneur ? » ou « comment fonctionnent les systèmes de contrôle de version ? ». Or, pour servir vos clients du mieux possible, ces questions ne sont pas moins importantes que votre compétence d'ordre purement technique. Nos clients nous demandent de nous occuper de l'aspect juridique des modèles économiques qui reposent sur une double licence de type « open core », mais aussi de la gestion des contrats de supports, de services, de développements ainsi que les conventions applicables à des contributions au code source. Nous

guidons nos clients pour leur indiquer quelles ressources pour leur entreprise ou institution peut leur offrir le logiciel libre et *open source*. D'autre part, nous proposons nos conseils aux développeurs pour qu'ils sachent comment réagir face aux violations de leur droit d'auteur, comment préparer et négocier des contrats pour leur compte. Afin de mieux servir ces clients globalement, il est important de bien connaître leurs différents points de vue.

Les connaissances en droit comparé

La troisième chose dont a besoin un juriste libriste, c'est de connaissances sur les juridictions étrangères, au moins quelques-unes, car plus il en acquiert, mieux il se porte. Pour pouvoir interpréter correctement les différentes licences, il est indispensable de comprendre le point de vue de ceux qui les ont élaborées. Dans la plupart des cas, le système juridique étatsunien est d'une importance capitale. La GPL par exemple a été conçue et rédigée avec, à l'esprit, des notions juridiques issues de ce pays. Aux États-Unis, le terme « distribution » englobe la distribution en ligne, tandis que le système de droit d'auteur allemand établit une distinction entre distribution en ligne et hors-ligne. Il s'ensuit que les licences rédigées par des juristes aux États-Unis peuvent être interprétées comme incluant la distribution en ligne. Au cours d'une procédure judiciaire, cet élément peut s'avérer pertinent et utile[1].

Apprendre en permanence

Au final, toutes ces connaissances sont d'une grande utilité. Aussi, de même qu'un logiciel est constamment maintenu et développé pour répondre aux besoins du moment, de même mon esprit continuera à chercher des réponses aux défis lancés aux juristes par la très active communauté du logiciel libre et *open source*.

---

1 http://www.ifross.org/Fremdartikel/LGMuenchenUrteil.pdf, Cf.Welte v.Skype, 2007.

42. Lancer des ponts

Shane Coughlan

Shane Coughlan est un expert en méthodes de communication et en développement d'affaires. Il est surtout connu pour créer des passerelles entre les différents domaines, commercial et non commercial, des technologies de l'information. Au plan professionnel, il a œuvré avec succès à la création d'un département juridique pour la Free Software Foundation Europe (FSFE), la principale organisation non gouvernementale pour la promotion du logiciel libre en Europe. Il a également contribué à l'établissement d'un réseau professionnel de plus de 270 experts juridiques et techniques à travers 27 pays, à la fondation d'un projet d'outil de conformité du code binaire. Il a travaillé à rapprocher intérêts privés et communautaires pour le lancement du premier examen critique d'une loi dédiée au logiciel libre et open source. *Shane a une connaissance étendue des technologies de l'Internet, des bonnes pratiques de gestion, de la construction de communautés et du logiciel libre et* open source.

Lorsque j'ai commencé à travailler dans le logiciel libre, j'ai été frappé par ce qui était perçu comme une différence entre les parties prenantes « communautaires » et « commerciales » dans ce domaine. Selon l'opinion largement admise à l'époque, les développeurs s'intéressaient au bidouillage du code et les commerciaux utilisaient leur travail de manière contestable s'ils n'étaient pas surveillés de près. C'était un préjugé relativement infondé et défendu presque exclusivement par des gens qui s'identifiaient à la communauté plutôt que par ceux qui se préoc-

cupaient davantage des intérêts commerciaux. Mais il était très répandu.

Bien que je sois principalement associé à la partie communautaire des choses, j’ai résisté à l’idée selon laquelle il y aurait deux camps intrinsèquement ennemis se faisant face à propos de l’avenir du logiciel libre. Cela semblait trop simple pour décrire les dynamiques de contribution, d’utilisation et de support comme une interaction entre les nobles créateurs et les parasites téléchargeurs de gratuit. Cela semblait en effet être plutôt une situation dans laquelle la complexité, les changements et l’incertitude avaient conduit à la création de récits simplistes destinés à rassurer un peu ceux qui sortaient de leur position confortable. Je pouvais ressentir la tension ambiante, entendre les querelles autour des stands lors des rencontres et les commentaires acerbes ou bien les montées en pression dans les conférences. Mais que signifiait tout cela ?

Que nous parlions de contribution à des projets de logiciels libres, de gestion de projet ou de respect des licences, les relations entre les parties prenantes s’accompagnaient souvent de préjugés, d’un manque de communication et d’émotions négatives. Ce qui conduisait ensuite à une plus grande complexité, à une augmentation proportionnelle de la difficulté à prendre des décisions unifiées ainsi qu’à résoudre les problèmes. Je savais que l’un des plus grands défis était de jeter des ponts entre les individus, les projets et les entreprises. C’est une étape nécessaire pour assurer une compréhension commune et une communication croisée des règles, des normes et des raisons qui sous-tendent les licences et autres mesures officielles qui régissent ce domaine. Mais le savoir ne suffit pas à trouver le moyen de s’attaquer efficacement au problème.

C’était la période charnière où la GPLv3[1] était en cours de rédaction. Les technologies fondées sur Linux commençaient à apparaître dans toutes sortes de produits électroniques grand

1 https://www.gnu.org/licenses/quick-guide-gplv3.fr.html.

public et le logiciel libre était sur le point de se démocratiser. Il y avait du changement dans l'air et les investissements commerciaux autour des principaux projets de logiciel libre atteignaient des sommets. Tout à coup, on voyait des employés de grandes entreprises s'atteler à des tâches complexes, on avait des fonds importants pour les évènements. De nombreux projets logiciels cessaient d'être une simple question de plaisir et commençaient à connaître les étapes planifiées, les livrables, l'assurance qualité et l'ergonomie.

Ce fut probablement un sérieux bouleversement pour ceux qui réalisaient du logiciel libre depuis longtemps : la majeure partie de l'évolution du logiciel libre ne concernait pas seulement l'exploration et la perfection technique, mais aussi l'interaction sociale. C'était un bon moyen pour des gens intelligents bien que parfois bizarres de partager un intérêt commun, de se lancer des défis et de coopérer au sein de projets bien définis et prévisibles. Tout comme collectionner des timbres, compter les trains ou connaître (par cœur) l'univers de Star Trek, c'était quelque chose qui fédérait des gens avec des centres d'intérêt bien particuliers en leur offrant de surcroît le bénéfice d'une agréable socialisation. Les premiers contributeurs ne s'attendaient sûrement pas à rencontrer là des cadres moyens et un développement orienté vers le volume de production. Pas étonnant que certains s'y soient cassé le nez.

Et pourtant… Tout s'est bien passé. Le logiciel libre est partout et sa position paraît être inexpugnable en tant que composant majeur de l'industrie des technologies de l'information. Des projets comme le noyau Linux ou le serveur web Apache ont continué à se développer, à innover et à attirer de nouvelles parties prenantes, tant commerciales que non commerciales. L'équilibre des pouvoirs entre les individus, les projets et les entreprises a changé, parfois avec des conflits et des perturbations, mais jamais au détriment d'une coopération sur le long

terme, ni en portant préjudice aux valeurs fondamentales du logiciel libre.

De mon point de vue, dans le domaine juridique – qui n'est après tout qu'un langage formel qui fournit un contexte pour l'interaction au travers de règles mutuellement comprises et applicables – la tension dans le logiciel libre ne s'est pas formée avec l'introduction d'une activité commerciale accrue, ni avec la participation grandissante d'employés dans des projets, ni avec le changement lui-même. Le vrai problème réside dans l'écart entre une élite précédente en perte de vitesse et de nouveaux venus parfois très différents.

Le défi consistait à élaborer des règles du jeu équitables avec lesquelles les différents intérêts pourraient coexister dans un respect mutuel. Le logiciel libre devait devenir un point de rencontre où n'importe qui pouvait à tout moment obtenir des informations comme les compétences appropriées, les obligations d'une licence ou les prérequis pour la soumission de code à un projet. La subjectivité et l'imprécision devaient être mises de côté pour permettre l'émergence de transactions plus formelles, qui deviennent comme le signal précurseur essentiel d'une forte activité économique, en particulier dans le contexte d'une communauté internationale voire mondiale.

Ce qui avait fonctionné dans les premiers jours – qu'il s'agisse de la confiance de quelques-uns ou de l'entente mutuelle d'un groupe homogène ayant des intérêts communs – ne pouvait plus agir en tant que facteur social ou économique pour l'avenir du domaine. Parfois, on avait l'impression que c'était une barrière insurmontable et que les tensions entre les contributeurs de longue date au logiciel libre et les nouveaux acteurs devaient conduire à un effondrement de la coopération et, peut-être, à celui des progrès accomplis. Mais un résultat aussi sombre supposait des conditions qui n'existaient tout simplement pas.

Le logiciel libre a apporté beaucoup de choses à toutes sortes d’individus et organisations, en se fondant sur quelques concepts très simples comme la liberté d’utiliser, de modifier, d’améliorer et de partager la technologie. Ces concepts ont permis beaucoup de flexibilité. Et tant que les gens reconnaissaient leur valeur et continuaient à les respecter, les conflits portant sur des points secondaires comme la gouvernance ou les zones d’ombre des licences étaient plutôt sans importance – à long terme. Le reste est principalement du bruit. Les communications habituelles sont dérangées par tous ces pièges dramatiques qui arrivent inévitablement lorsqu’un groupe social est rejoint par un autre. Cela s’applique à toutes les discussions, que l’on parle d’un bon coin de pêche, d’un pays qui accueille (ou non) les immigrants ou de deux entreprises qui fusionnent.

Les changements au sein du logiciel libre paraissaient tous un peu perturbants à l’époque, mais on peut en dégager principalement trois leçons utiles qui seront familières aux étudiants d’histoire ou de sciences politiques. Premièrement, à chaque fois qu’il existe une élite, elle cherchera à préserver son statut et décrira le conflit perçu en termes d’évolution négative dans le but de l’ébranler. Deuxièmement, malgré la tendance inhérente de chaque base de pouvoir à être conservatrice, la participation statique dans un domaine en évolution aura pour seul effet de déplacer la possibilité d’une amélioration des parties existantes vers de tierces parties. Enfin, si quelque chose a de la valeur, alors les difficultés rencontrées en matière de gouvernance sont peu susceptibles de porter atteinte à cette valeur, mais fourniront au contraire une méthode pour redéfinir à la fois les mécanismes de gouvernance et les personnes en mesure de les appliquer.

Le développement du logiciel libre en tant que technologie démocratisée a connu une professionnalisation croissante parmi les développeurs comme dans la gestion de projets. Nous avons aussi vu s’accroître le respect envers les licences de la part des individus, des projets et des entreprises. Ce ne fut pas une

mauvaise chose, et malgré quelques accidents de terrain en cours de route – au choix : les querelles inter-communautaires, les entreprises qui ne tiennent pas compte des termes des licences ou l'agacement causé par un éloignement de l'esprit décontracté habituel – notre position en est consolidée, plus cohérente et de plus grande valeur.

Postface

De *Open Advice* aux *Libres conseils*

Débuté en décembre 2012, ce projet a mobilisé plus de 90 contributeurs autour d'un but commun, permettant à tous de mettre à profit leurs talents et leur envie de participer à un projet libre et collaboratif. Venus de Twitter, de LinuxFR[1], anciens ou nouveaux de Framalang[2], tous étaient réunis pour aboutir au livre que vous venez de parcourir.

Cette masse conséquente de contributeurs a été encadrée par un petit groupe de Framalinguistes. Tétanisés par une organisation millimétrée, ils ont orchestré la traduction des chapitres, leur publication sur le Framablog[3], leur relecture attentive et enfin la publication tant attendue de cet ouvrage. Les annonces et invitations sur LinuxFr ont permis de recruter de nouveaux traducteurs aux personnalités fort différentes qui apportaient chacun leur touche à l'ouvrage. Le travail a progressé pour parvenir à présent à son terme. Ce chapitre constitue le récapitulatif de cette passion-

1 http://linuxfr.org.
2 http://www.framalang.org.
3 http://www.framablog.org.

nante période, parsemée de rencontres, de nouvelles amitiés, et de la satisfaction du travail accompli ensemble.

Réparti sur différents supports, des *pads* collaboratifs basés sur Etherpad à la plateforme d'édition Booktype, ce projet est la démonstration par l'exemple que chacun peut participer à un projet libre, quelles que soient ses compétences. Les outils utilisés sont sous licence libre, dans l'esprit de Framasoft, et permettent donc leur réappropriation ultérieure.

La naissance d'une idée

Open Advice, version anglaise de ce livre, a été proposé au public en janvier 2012. Véritable trésor pour ceux qui veulent se lancer dans le logiciel libre, il n'était pas disponible en français. Quelques mois plus tard, LinuxFR, qui avait déjà croisé la route de l'association Framasoft dans le cadre de traductions d'articles, proposa un partenariat pour monter un projet ambitieux : la traduction collaborative de cet ouvrage qui comptait pas moins de 310 pages. Toute la difficulté résidait dans la mise en œuvre tant des outils de traduction collaborative que de méthodes adaptées pour permettre au projet de ne pas s'essouffler en cours de route, et d'être mené à son terme. C'est là que l'expérience de Framasoft, à travers son groupe de traduction Framalang, pouvait jouer tout son rôle.

Des séances de traductions cadencées

C'est à la mi-décembre 2012 que le projet a officiellement démarré. Invités par trois animateurs du projet, des traducteurs de tous horizons se sont donné rendez-vous un soir par semaine, à heure fixe, pour traduire ensemble quelques chapitres du recueil de conseils, dans une ambiance plus ou moins studieuse, mais toujours débordante de bonne humeur. De nombreux participants ont rapidement répondu aux appels lancés sur LinuxFR, sur le blog de Framasoft ou encore sur les réseaux sociaux, venant renforcer les traducteurs réguliers de FramaLang. Et c'est très vite

un groupe régulier qui, de session en session, a rondement rempli la lourde tâche qui consistait à traduire au plus juste les différents conseils prodigués par ces acteurs émérites du monde du logiciel libre. Entre échanges vaches et collaboration intensive, ce sont près de 90 personnes qui se sont associées pour permettre à cette traduction d'avancer de façon régulière.

Une relecture consciencieuse

Pour donner suite aux traductions sorties en soirée de toute l'équipe de traducteurs, un petit groupe composé des animateurs et de quelques fidèles contributeurs à l'œil affûté a eu pour tâche de réviser, relire, corriger, remanier, torturer les mots afin de rendre le tout publiable tout d'abord sur le Framablog.

Une deuxième relecture a suivi, prenant en compte les remarques souvent pertinentes des lecteurs du blog au sujet de remaniements/corrections à apporter pour une œuvre de plus grande qualité. Un petit groupe d'irréductibles relecteurs et relectrices (*grammar-nazi* pour les intimes) a ensuite eu l'immense privilège de s'en donner à cœur joie pour rétablir espaces insécables, guillemets français et autres apostrophes typographiques qui étaient passés à travers les mailles du filet lors de la première révision.

Cette nouvelle phase de travail était aussi l'occasion de retravailler à volonté certains passages afin de les rendre plus fluides à la lecture dans l'espoir de satisfaire au mieux le lecteur de cet ouvrage. Travail extrêmement chronophage, cette phase a duré presque aussi longtemps que les séances de traduction, si ce n'est plus. Cette tâche est beaucoup plus ingrate car souvent invisible et sous-estimée, autant par le lecteur final que par les traducteurs eux-mêmes. Les participants se retrouvent plus isolés, seuls face aux relectures et il est beaucoup plus difficile de garder le rythme. La motivation est également difficile à maintenir, car le résultat d'heures de travail passées à relire les mêmes paragraphes est beaucoup moins visible.

De la relecture à la publication

Le livre était alors prêt à entrer dans sa dernière phase (la mise en page) et à rejoindre les mains expertes de l'équipe des bénévoles de Framabook. Élaboration de la couverture, rédaction de la quatrième de couverture, mise en forme de l'ensemble du texte selon la maquette Framabook, dans le respect des standards de l'édition papier comme de la future édition numérique… *Libres conseils* commençait à ressembler à ce livre que vous tenez entre les mains. Il ne restait plus qu'à l'inspecter sous toutes les coutures pour repérer les erreurs de mise en pages ou corriger les quelques coquilles qui auraient pu échapper jusque-là à la vigilance des relecteurs. À ce stade, le changement de support et le renfort de personnes extérieures au projet est bienvenu car il permet de les repérer plus facilement, et d'améliorer ainsi la qualité du recueil final.

Un livre, une expérience fructueuse

Le projet de traduction de ce livre aura finalement pris presque une année complète, de la phase de préparation à la publication. Avec un nombre de contributeurs de tous horizons particulièrement important, il démontre que chacun peut participer à un projet collaboratif libre, avec une ambiance conviviale, quelles que soient ses compétences. Cela illustre parfaitement le contenu de ce recueil. Le logiciel libre ne se limite pas au seul code, et il est tout à fait possible à chacun de participer à la hauteur de ce qu'il veut et sait faire. Ne vous interdisez pas de rejoindre des projets libres, quels qu'ils soient, sous prétexte que vous ne pourriez pas être utiles car vous ne savez pas coder, ou parce que vous ne maîtrisez pas le sujet. Sur les projets, toutes les compétences sont importantes. Il faut bien un début à toute chose, et vous trouverez toujours d'autres contributeurs pour vous guider dans vos premiers pas.

Et maintenant, à vous de jouer…

Vous pouvez prolonger cette expérience et lui apporter votre contribution. Bien entendu nous serions ravis si votre approche du Libre et de l'*open source* s'en trouve éclairée et enrichie.

Mais vous pouvez aussi contribuer de diverses façons. Et ici nous ne résistons pas au plaisir de dresser une dernière liste.

- Si la lecture vous a été utile, vous pouvez faire connaître ce livre autour de vous, en recommander la lecture, y faire référence et en partager les contenus.
- Vous pouvez aussi nous signaler les petites et grosses erreurs qui malgré toute notre attention peuvent nous avoir échappé : vous améliorerez ainsi les éditions suivantes.
- Et qui vous empêche d'écrire à votre tour le récit de vos premières armes dans le Libre et l'*open source* ? Nous pensons que l'expérience doit être partagée, alors n'hésitez pas à rejoindre *Libres conseils*…

Pour nous contacter : http://framabook.org/nous-contacter

Crédits

Plus qu'un simple recueil de conseils sur les logiciels libres, cet ouvrage est aussi une véritable aventure humaine pour ceux qui y ont contribué. Initialement créé par une coordinatrice et 42 auteurs, il a emporté avec lui de nombreux autres contributeurs sur cette version française.

Une communauté de traducteurs

Aa, AlBahtaar, Alexb, Alpha, Antoine, Arthur, Audionuma, Benoît Audouard (Baud), Jean-Noël Avila, Laurent Bergere (Kalupa), Billouche, Quentin Binot (4nti7rust), Bob, Barbara Bourdelles (Garburst), Gatien Bovyn (Astalaseven), JF Carrasco, Sébastien Chauvel (Sinma), Thomas Citharet (tcit), Coco, Cédric Corazza, Corentin, CoudCoud, Cyrille L, Dabou, Luc Didry (Sky), e-Jim , Elquan, Erdesc, Ex0artefact, Eyome, Floxy, Fred, Freepius, Fubik, ga3lig, Gilles, GPif, Gregseth, grosfar, Sandra Guigonis (peupleLà), HanX, Julien Henry (Julius22), HgO, Hideki, Husi10, jcr83, Jej, Kev, Khyvodul, KoS, LAuX, lenod, lerouge, Liar, Lignusta, LuD-up, Lum', Lycoris, maat, Jean-Bernard Marcon (Goofy), maxlath, Merlin 8282, meumeul, Michel, Miki, Munrek, noskill, Nys, Okram, Nicolas Ourceau (Lamessen), Peekmo, Chloé Philippe (Ouve), purplepsycho,

Liu Qihao (Eastwind), RavageJo, Sasha_01, satanas_g, schap2, Slystone, Sphinx, Sputnik, Sylvain, Tala, Tsigorf, Tuki, Vilnus Atyx, vvision, Wxcafe, Yann, zn01wr.

Une équipe de relecteurs

Gatien Bovyn (Astalaseven), Sandra Guigonis (peupleLà), Julien Henry (Julius22), Jean-Bernard Marcon (Goofy), Christophe Masutti (Framatophe), Nicolas Ourceau (Lamessen), Barbara Bourdelles (Garburst)

Une équipe d'animateurs/coordinateurs

Sandra Guigonis (peupleLà), Jean-Bernard Marcon (Goofy), Nicolas Ourceau (Lamessen)

Annexe

Lexique

*Que serait ce livre sans l'explication de quelques termes employés par ses auteurs ? Les définitions contenues dans ce lexique sont volontairement succinctes. Elles sont majoritairement issues de l'*encyclopédie Wikipédia*. N'hésitez pas à la consulter pour plus d'informations.*

A

API *(Application Programming Interface)* : une interface de programmation est une façade formalisée pour laquelle un logiciel offre des services à un autre logiciel.

ARM : famille de processeurs à l'architecture simplifiée, très utilisés dans les systèmes embarqués, les smartphones et les tablettes.

B

Beamer : classe LaTeX adaptée à la création de présentations.

bibliothèque (logicielle) : collection de sous-programmes compilés prête à être utilisée par d'autres programmes.

brainstorming : technique de groupe utilisée dans la résolution de problèmes qui consiste à produire un maximum d'idée sur un thème fixé, sans y apporter de critiques dans un premier temps.

bugsquad : équipe dédiée à la résolution des bogues.

bugtracker : logiciel de suivi des problèmes.

C

codebase : terme utilisé en développement logiciel pour désigner l'ensemble du code source utilisé pour construire un logiciel.

code infectieux : Équivalent approché du terme bitrot qui en argot de codeur désigne ce fait quasi-universel : si un bout de code ne change pas mais que tout repose sur lui, il « pourrit ». Il y a alors habituellement très peu de chances pour qu'il fonctionne tant qu'aucune modification ne sera apportée pour l'adapter à de nouveaux logiciels ou nouveaux matériels.

commit : action d'envoyer ses modifications locales vers le référentiel central.

CVS *(Concurrent Version Systems)* : Le système de gestion de versions, créé en 1990, a été largement utilisé par les projets de logiciels libres. Le modèle de CVS est un modèle centralisé, où un serveur central regroupe toutes les sources.

D

diff : fichier qui affiche le résultat d'une comparaison entre deux éléments (en général, des lignes de code).

DNS *(Domain Name System)* : service permettant de traduire un nom de domaine en informations de plusieurs types qui y sont associées, notamment en adresses IP de la machine.

DVCS *(Distributed Version Control System)* : la gestion de versions décentralisée est un outil de gestion de versions qui permet de travailler sur divers versions simultanément, de façon désynchronisée des autres, puis d'offrir un moyen à ces développeurs de s'échanger leur travaux respectifs. De fait, il existe

plusieurs dépôts pour un même logiciel. Ce système est très utilisé par les logiciels libres.

F

fork : logiciel créé à partir du code source d'un autre logiciel. Cela nécessite que les droits accordés par les auteurs du logiciel initial le permettent. C'est pourquoi les *forks* se retrouves principalement dans les logiciels libres.

FHS *(Filesystem Hierarchy Standard)* : la « Norme de la hiérarchie des systèmes de fichiers » définit l'arborescence et le contenu des principaux répertoires des systèmes de fichiers des systèmes d'exploitation GNU/Linux et de la plupart des systèmes Unix.

G

git pull : commande git qui permet de mettre à jour un dépôt local et s'assurer qu'elle correspond à la dernière version des fichiers sur le serveur principal.

I

IRL *(Internet Relay Chat)* : discussion relayée par Internet. Protocole de communication textuelle sur Internet.

L

localisation : processus d'adaptation d'un produit logiciel ou documentaire à une région donnée. Cela comprend la traduction dans la langue de la région mais aussi l'adaptation aux normes, à la culture et aux besoins spécifiques de cette région du monde.

ligne commutée : réseau qui servait aux communications téléphoniques à la fin du XXe siècle, basé sur une paire torsadée de fils en cuivre. L'accès à Internet grâce à ce réseau se fait à l'aide

d'un modem téléphonique. C'est par ce réseau qu'ont été commercialisées, en France, les premières offres internet.

MD4 *(Message Digest 4)* : algorithme de hachage conçu en 1990, qui propose une signature de 128 bits. L'algorithme a été abandonné au profit du MD5 après la découverte de faiblesses dans sa conception (Den Boer et Bosselaers).

miroir : copie exacte d'un ensemble de données. Les dépôts miroirs permettent de fournir plusieurs copies de la même information, répartissant ainsi la charge générée par un trafic élevé sur plusieurs serveurs.

modem *(modulateur-démodulateur)* : périphérique servant à communiquer avec des utilisateurs distants par l'intermédiaire d'un réseau analogique (comme une ligne téléphonique). Il permet par exemple de se connecter à Internet.

N

NTIC *(New Information and communication technologies)* : les nouvelles technologies de l'information et de la communication regroupent les techniques utilisées dans le traitement et la transmission des informations, principalement de l'informatique, de l'internet et des télécommunications.

NTLM *(NT Lan Manager)* : protocole d'identification utilisé dans diverses implémentations des protocoles réseau Microsoft.

P

PHP *(Hypertext Preprocessor)* : langage de programmation compilé principalement utilisé pour produire des pages web dynamiques.

R

rsync (remote synchronization) : « synchronisation distante » est un logiciel libre de synchronisation de fichiers, distribué sous licence GNU GPL. La synchronisation est unidirectionnelle, c'est-à-dire qu'elle copie les fichiers de la source en direction de la destination.

Refactorisation : bien que critiqué, cet anglicisme semble d'un emploi courant chez les développeurs. On pourrait parler plus simplement de remaniement.

release : publication d'un logiciel, sa mise à la disposition du public.

S

skill : « compétences ». Ce terme anglais est utilisé sans traduction dans le monde du jeu vidéo, y compris dans les pays francophones.

U

UML *(Unified Modeling Language)* : le « langage de modélisation unifié » est un langage de modélisation graphique à base de pictogrammes.

V

VPS *(Virtual Private Server)* : un « serveur privé virtuel » est une méthode de partitionnement d'un serveur en plusieurs serveurs virtuels indépendants qui ont chacun les caractéristiques d'un serveur dédié, en utilisant des techniques de virtualisation. Cela permet notamment de faire fonctionner chaque serveur avec un système d'exploitation différent, ou redémarrer les serveurs indépendamment les uns des autres.

X

X.org : serveur X libre. Il fonctionne sur la plupart des systèmes d'exploitation de type UNIX et est très populaire au sein de la communauté du logiciel libre. Il tend toutefois à être remplacé par Wayland (ou MIR).

Table des matières